For these reasons, PSI respectfully suggests that the term "target routine" is more than a mere set, and includes the concept of a sequence as set forth in PSI's proposed construction.

b. What does a target routine do?

The other dispute between the parties concerns the *function* of a target routine. PSI believes that a target routine "performs a function similar to, but not necessarily identically to, a corresponding incompatible instruction," while IBM asks the Court to give it the broader function of "enabling an incompatible instruction to be run on target computer system."

This dispute should be no contest. PSI took its definition of what the target routine does ***directly*** from the claims at issue. Claim 1, for example, says explicitly that "each target routine perform[s] a function similar to, but not necessarily identically to, a corresponding incompatible instruction." '261 at 19:35-37. In other words, PSI's description of what a target routine *does* comes directly from the claims at issue.

IBM's proposal, on the other hand, is both confusing and imprecise. What does it mean to say, as IBM does, that a target instruction "enables" an incompatible instruction to be "run on" the target computer system? Given this imprecision, under IBM's constructions there is no difference between a "semantic routine" and a "target routine." Consider:

IBM proposal for "semantic routine"	**IBM proposal for "target routine"**
A defined set or sequence of native instructions that, when executed, emulates an associated guest instruction.	The set or sequence of target instructions that enable an incompatible instruction to be run on target computer system.

Under IBM's constructions all target routines qualify as semantic routines and all semantic routines qualify as target routines. But, while target routines can *in some circumstances* be used as semantic routines, the whole point of the '261 patent is that you save memory space by storing target routines that are patched dynamically ***rather than*** complete semantic routines that are translated in advance. *See* '261 at 3:67-4:4 ("by doing the instruction translation in a dynamic

manner each time an incompatible instruction is encountered, it is unnecessary to save the translations for future use."); 5:18-28 ("The large amount of target storage required to save the existing translations for future execution and for the incompatible instruction directory is saved by the dynamic translation of incompatible instructions as they are encountered during emulated execution."). To put it another way, "target routines" are *templates* that can be used to create routines of target instructions which, when executed, emulate an incompatible instruction. *See* '261 at 5:8-13 ("The subject invention described here uses an emulation preprocessing function, which decodes the incompatible instructions and, for each, determines the location of a target processor translation *template routine* that performs the function of the incompatible instruction. A template routine is executed by the processor..."). Thus, the definitions of the terms "semantic routine" and "target routine" should reflect (as PSI's proposals do) that some target routines are semantic routines, but should not (like IBM's constructions) conflate the two terms completely.

IBM's only argument to the contrary relies not on intrinsic evidence but, instead, on a purported point of grammar. Specifically, IBM argues that PSI's proposal should be rejected because PSI's language about what the target routine does is taken straight from the claim, and thus replacing the phrase "target instruction" with PSI's construction results in repetition. IBM Br. at 50. This is a truly bizarre argument. IBM does not cite a *single* case (and PSI is aware of none) standing for the proposition that the way to tell whether a construction is correct is to swap the construction in for the term it defines and see if the thus-modified sentence is still grammatical. Despite the existence of thousands of published Federal Circuit opinions, not one supports a "does the grammar still work?" test.

IBM's own brief establishes that it does not take this Grammar Test seriously, since elsewhere in its brief IBM acknowledges that the claims "are accorded the most weight" in claim construction. IBM Br. at 64. IBM's construction of "floating point unit" illustrates the point. As

discussed *infra* in Section VI.A.1, IBM construes "floating point unit" in the '106 patent as "part of a computer system that performs floating point operations." *Id.* at 60. IBM defends the "part of a computer system..." portion of its construction by noting that claim 11 of the '106 patent begins with "[i]n a computer system, a floating point unit that...." *Id.* at 60-61 (citing '106 patent at 22:7). But this argument ignores the Grammar Test, and that construction flunks it, because when you plug in IBM's construction claim 11 reads as follows: "In a computer system, a portion of a computer system that...." In short, the Grammar Test is silly, is not supported by any authority, and is something IBM only uses when it cannot grapple with the intrinsic evidence.

PSI's construction should be adopted.

VI. THE FLOATING POINT PATENTS

The '106, '678 and '709 patents all relate to a numerical representation system known as "floating point," which is commonly used in computers. In short, floating point is a system in which a string of digits (*i.e.*, 0s and 1s) represents a real number. This system is called "floating" because the radix point (what humans typically call the "decimal point," and what computer engineers typically call the "binary point") can "float": that is, the radix point can be placed anywhere relative to the significant digits of the number. The advantage of floating-point representation over fixed-point representation is that it can convey a much wider range of values. For example, a fixed-point representation that has eight decimal digits, with the decimal point assumed to be positioned after the sixth digit, can represent the numbers 123456.78, 8765.43, 123.00 and so on, whereas a floating-point representation with eight decimal digits could also represent 1.2345678, 1234567.8, 0.000012345678, 12345678000000000000, and so on.

Since the 1950s, different computers have utilized different floating-point systems. Two standards, however, became dominant. Beginning in 1964, IBM began using the hexadecimal floating point ("HFP") format, which is characterized by its use of a base of 16 (humans

typically think in terms of the decimal system, which uses a base of 10). In 1985, the IEEE released a standard known as the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985). Binary floating point ("BFP") format, which uses a base of 2, is much more common in computer systems than HFP, and since its release the IEEE's 754-1985 standard has become by far the most commonly used standard.[27]

By the mid-1990s, IBM felt compelled to make its HFP-based systems compatible with the increasingly popular IEEE 754-1985 BFP format. '106 patent at 1:22-45. The '106 patent, entitled "Implementation of Binary Floating Point Using Hexadecimal Floating Point Unit," claims a computer system that has a "floating point unit" which is capable of "executing floating point operations in either IBM S/390 hexadecimal format or IEEE 754 binary format." '106 patent at Abstract. The '678 patent notes that "future machines," like the one disclosed in the '106 patent, "must be compatible with previous machines and must also provide support for new [floating point] formats. Thus, a new requirement emerges to provide hardware support for more than one format." '678 patent at 1:32-36. The '678 patent claims a "floating point processor" capable of providing this hardware support—in particular, a floating point processor capable of executing an instruction "that will provide more complete information on the data class of a floating point number." *Id.* at 1:49-51. Finally, the '709 patent also provides hardware support for the new HFP and BFP compatible machines IBM was building "by providing a machine having a default rounding mode that may be overridden by a rounding mode designated by an instruction." '709 patent at 1:40-42.

The '106, '678 and '709 patents were all filed on the same day, March 31, 1995. All three patents incorporate the following two documents into their specifications by reference:

[27] The Intel processors in PSI's machines are all EPIC architecture processors, and they therefore all utilize BFP and in particular the IEEE 754-1985 version of BFP. The System-z processors in IBM's mainframes are all IBM System-z architecture processors, and they therefore all utilize HFP.

> (1) The IEEE 754-1985 standard, which is the source of the floating-point "data classes" in the '678 patent and of the floating-point "rounding modes" in the '709 patent.
>
> (2) The 1994 edition of the *IBM ESA/390 Principles of Operation* (*i.e.*, the roadmap of the computer architecture for IBM's System 390 mainframes, which at the time were IBM's flagship system). This document is referred to throughout this brief as the "390 POO."

See '106 patent at 1:15-20, '678 patent at 1:14-20, and '709 patent at 1:15-20. In addition, the '678 and the '709 patents each incorporate the patent application for the '106 patent into their specifications. '678 patent at 1:22-26 and '790 patent at 1:22-25.

A. The '106 "Binary Floating Point Using Hexadecimal Floating Point Unit" Patent

By the time the '106 patent was filed in March 1995, the IEEE's 754-1985 BFP standard had become far and away the dominant floating-point standard used by computer architects. IBM had used its own HFP standard for decades, but, as the '106 specification tell us, IBM felt a need to accommodate users' demand for programs compatible with the IEEE 754-1985 standard:

> Generally, computer systems are designed in accordance with supporting a specific floating point architecture, such as either IEEE Binary Floating Point Standard 754-1985 or IBM S/390 hexadecimal floating point format. Generally, applications which are directed to one floating point standard are not compatible with another floating point standard. Thus, a user is limited in terms of "buying into" a certain architecture and to software applications developed therefor. Users, however, generally wish to run the same application on different platforms supporting different floating point architectures, or use data generated by applications on one platform with a different application on another platform that supports a different floating point architecture. Moreover, a user generally wishes to be able to run on a single machine applications that are based on different floating point architectures....There is a need, therefore, to have a computer system that supports two floating [point] architectures.

'106 patent at 1:22-45. One way to support two floating point architectures would be to simply have two floating point units. That, however, would be inefficient (it would cost more and take up more space), and it would not be much of an invention—BFP floating point units and HFP floating point units have been around for decades, and there's nothing inventive about simply installing both types of units in one machine. Instead, the invention of the '106 patent is a ***single***

"floating point unit" that allows you to support two floating point architectures "without having to essentially duplicate floating point hardware." *Id.* at 1:49-50 and all claims of the '106 patent, each of which claims a "floating point unit"). Continuing from where the block quote above left off, the specification states as follows:

> Moreover, such a computer system should provide the two floating point architectures with high performance and with limited resources. That is, multiple floating point architecture support should be provided **without having to essentially duplicate floating point hardware** and without sacrificing performance.

Id. 1:45-50 (emphasis added). Reflecting the nature of this invention, the longest part of the '106 specification is a section entitled "Overview of Hardware Support for Multiple Architectures." *Id.* at 1:46-47. That section discloses "[t]he actual hardware needed to support both architectures," including "4 types of format conversion hardware" and "the Floating Point Unit." *Id.* at 1:49-54.

The parties dispute four terms in the '106 claims: "floating point unit"; "a floating point unit having an internal data flow"; "supports both said first floating point architecture and said second floating point architecture"; and "converter."

1. "floating point unit" (claims 11 and 12)

IBM's Proposed Construction	PSI's Proposed Construction
Part of a computer system that performs floating point operations.	That portion of a processor's circuitry that performs floating point calculations.

There are two issues presented by this claim term:

(1) Whether all that can be said about a "floating point unit" is that it is "part of a computer system" (IBM's position, for this term and also for "processor," "floating point processor," and "converter"), or whether "floating point unit" has the more concrete meaning of a "portion of a processor's circuitry."

(2) Whether a "floating point unit" performs floating point *operations* or floating point *calculations*.

The second issue is simpler: there is no material difference between "operations" and "calculations," and so long as no improper meaning is imputed to "operations" PSI has no objection to the Court adopting that term as part of its construction. PSI believes *calculations* is more precise because all of the floating point operations in the '106 patent are in fact mathematical calculations (*see* '106 patent at 2:30- 20:46; Patt Decl. ¶ 23), but to simplify the issues PSI will accept "operations."

The remaining issue is whether a "floating point unit" is circuitry (*i.e.*, processing hardware) or whether a "floating point unit" cannot be described any more precisely other than to say that it is "a part of a computer system." The specification overwhelmingly demonstrates that a "floating point unit" is hardware circuitry. First, the specification makes clear that *the invention itself* is a single piece of hardware (*i.e.*, a floating point unit, or as it is called in the title of the patent, a "Hexadecimal Floating Point Unit") that is capable of supporting two floating point architectures and that thereby avoids the problem of "having to essentially duplicate floating point hardware":

> There is a need, therefore, to have a computer system that supports two floating [point] architectures. Moreover, such a computer system should provide the two floating point architectures with high performance and with limited resources. That is, multiple floating point architecture support should be provided **without having to essentially duplicate floating point hardware** and **without sacrificing performance**.

'106 patent at 1:44-50 (emphasis added).

The "without sacrificing performance" aspect of the '106 invention is also significant. Certain parts of a computer system are active and therefore have measureable "performance" (*e.g.*, a processor's performance is measured by its clockspeed, or in other words the number of instructions it executes per second), while other parts of a computer system are passive and therefore have no measureable "performance" (*e.g.*, software, which just sits there unless and until a human directs a processor to execute it). The specification makes clear that the *invention*

itself of the '106 patent is a "floating point unit" with "very high performance" on hex and binary operations:

> The present invention provides a floating point unit having a very high performance on hex operations and provides compatibility of binary operations, without significant loss of performance for these binary operations.

Id. at 20:43-46. There are numerous "parts of a computer system" (*e.g.*, software) that cannot experience a "loss of performance" and that therefore should not be included in the construction of "floating point unit." Instead, a "floating point unit" is clearly a piece of hardware that, in the case of the '106 patent, provides superior performance vis a vis the prior art.

The prosecution history makes this same point. In responding to the examiner's initial rejection of their claims, the inventors distinguished their invention over the prior art by noting that it was capable of converting between different *architectures*, while the prior art was only capable of converting between different formats within the *same architecture*. The inventors described the relevance of this distinction to the examiner as follows:

> This distinction may be appreciated by considering the distinctions between IBM S/390 hexadecimal floating point and IEEE 754 Binary floating point: each of these architectures inherently requires and prescribes *distinct floating point dataflows*. It is appreciated that converting between architected types has a more significant impact on ***hardware (and concomitantly chip real estate) requirements*** and further, requires consideration of performance/***hardware*** tradeoffs/optimization with respect to each architected type.

See Applicants' Response dated December 11, 1996 at 15 (Ex. 28) (emphasis added).

Consistent with their definition of the invention, the inventors' use of "floating point unit" in the patent clearly refers to a hardware device, and not to any software programs. The principal (and longest) section of the specification is entitled "Overview of Hardware Support for Multiple Architectures." '106 patent at 3:46-47. **All 17 of the claims in the patent claim a "floating point unit" that *"supports"* multiple floating point architectures**. *Id.* at 20:57-

22:65. The specification describes "[t]he actual hardware ***needed to support*** both architectures," making clear that a "floating point unit" is circuitry in a processor:

> **The actual hardware needed to support both architectures** consists of 4 types of format conversion hardware, a rounder, sticky bit determination, and controls for special number handling. FIG. 1 provides an overview of the dataflow of the Floating Point Unit (FPU) which supports both S/390 and IEEE 754 architecture. **The FPU has a localized high-speed memory**

The extrinsic evidence also makes clear that a "floating point unit" is processing circuitry:

1. Numerous technical dictionaries agree that a "floating point unit" is *circuitry*, not merely some combination of hardware and software that performs floating point operations. The *Comprehensive Dictionary of Electrical Engineering* defines a "floating point unit" as "a ***circuit*** that performs floating-point computations, which is generally addition, subtraction, multiplication, or division." (Ex. 29.) The *Microsoft Computer Dictionary* defines a "floating point unit" as "[a] ***circuit*** that performs floating-point calculations." (Ex. 30.) IBM itself defines a "unit" as "[a] mechanical, electrical, or electronic piece of equipment for a special purpose." http://www-306.ibm.com/software/globalization/terminology/tu.jsp#x2042528 (Ex. 31).
2. *See* E. M. Schwarz, L. Sigal, and T. J. McPherson, "CMOS floating-point unit for the S/390 Parallel Enterprise Server G4," *IBM Journal of Research and Development*, Vol. 41, No. 4/5, pp. 475-488 (1997), which discusses the "S/390 floating point unit (FPU)," describes the FPU's various "circuits," and notes that the FPU is "5.3 by 4.7 mm in size." (Ex. 32.) One of the authors of this article, E.M. Schwarz, is a named inventor of U.S. Pat. No. 6,009,261 and U.S. Pat. No. 5,687,106, two of the patents-in-suit.
3. *See* U.S. Pat. No. 7,117,389 (filed Sept. 18, 2003) (Ex. 5), assigned to IBM, which has a "Background" section discussing "the information revolution" and in particular the development of the "central processing unit (CPU) and supporting hardware," and noting the following: "Floating point calculations are among the most time-consuming instructions for processors to perform…[M]ost modern high-performance processors have some form of special floating point functional unit for performing

> floating point calculations. Due to the complexity of floating point calculations, a floating point unit is generally implemented as a pipeline. Such a pipeline typically requires a significant area of an integrated circuit chip, composed primarily of custom logic. Even though floating point operations are not initiated in every cycle, the circuits within a floating point pipeline have a high degree of switching activity, and consume considerable power at the operating frequencies typical of such devices." *Id.* at 1:24&33, 2:35-48.

IBM offers three arguments to support its "part of a computer system" argument. First, IBM points to claim 11 and notes that it contains the following phrase: "In a computer system, a floating point unit that...." According to IBM, this phrase indicates that a floating point unit should be construed as being nothing more than "part of a computer system." IBM Br. at 61. This argument lacks merit. The introductory phrase "in a computer system" is nothing more than that—an introductory phrase. That phrase does not indicate that everything after it should be construed to be only "part of a computer system," and it does not even remotely suggest that a FPU can include "software."

IBM's second argument is that the '106 inventors stated at the end of the specification that "the present invention is not limited to the disclosed embodiments." IBM Br. at 61. This language is nothing more than boilerplate verbiage that is included in nearly all patents issued by the PTO, including all 10 of the patents-in-suit.[28] This boilerplate language has no relevance to how the Court should construe "floating point unit."

Finally, IBM contends that the terms "processor" and "circuitry" do not appear anywhere in the specification. *Id.* at 61-62. That assertion is simply wrong: **the terms "processor" and**

[28] *See* '106 patent at 20:47-56; '678 patent at 4:54-58; '709 patent at 4:38-48; '789 patent at 15:11-17; '495 patent at 11:9-19; '261 patent at 19:14-20; '520 patent at 16:1-9; '002 patent at 8:19-30; '812 patent at 10:66-11:4; and '851 patent at 31:65-32:2.

"circuitry" appear a total of *167 times* in the specification. Though IBM was apparently unaware of this fact, the '106 specification incorporates by reference and therefore includes the 1994 edition of the System 390 Principles of Operation (the "390 POO"). *See* '106 patent at 1:13-17. The 390 POO refers repeatedly to both "processor" and "circuitry."[29] Among other things, the 390 POO makes clear that "processors" are hardware circuitry and not, as IBM disingenuously contends in this lawsuit, amorphous objects that could include monitors, keyboards and the like. *See, e.g.*, 390 POO at 2-2 & 2-3 (discussing "[t]he physical implementation of the CPU," "the width of the processing elements," and the fact that the "physical storage medium" which the CPU uses to "perform its functions" is "not considered part of main storage," but instead consists of the CPU's own "PSW" and "registers"). Furthermore, the number of times the word "circuitry" appears in the specification is irrelevant: "circuitry" refers broadly to processing hardware, and as discussed above that type of hardware is replete throughout the specification. IBM's mistaken assertion about the number of occurrences of "processor" and "circuitry" in the specification does not support IBM's "part of a computer system" construction.

In sum, IBM's amorphous construction is contrary to the intrinsic and extrinsic evidence and should be rejected.

2. "a floating point unit having an internal dataflow" (claim 11)

IBM's Proposed Construction	PSI's Proposed Construction
A floating point unit which includes the functions of moving data through the unit and executing floating point operations.	A floating point unit that uses an internal floating point format to perform floating point calculations.

The parties' respective contentions concerning "floating point unit" and "operations" versus "calculations" was discussed above. The only issue presented by this claim term is

[29] A copy of the relevant pages of the 600-page 390 POO is attached to the Nagy Decl. as Exhibit 8.

whether "having an internal dataflow" should be construed to mean "includes the functions of moving data through the unit and executing floating point operations" (IBM's construction) or "uses an internal floating point format to perform floating point calculations" (PSI's construction).

The intrinsic evidence makes clear that the ***internal*** dataflow of the FPU disclosed in the '106 patent utilizes an ***internal*** floating point format. Specifically, claim 11 claims "a floating point unit having an internal dataflow ***with an internal floating point format*** which supports both" floating point architectures. *Id.* at 22:11-12 (emphasis added). The very essence of the FPU claimed in claim 11 is that it uses an internal floating point format (referred to as "Hex-I" in the specification—*see, e.g.*, Figure 1) to perform all of the floating point operations for the two floating point architectures that this FPU supports:

> HEX Internal Dataflow
>
> In accordance with the present invention, providing support for two floating point architectures may be provided by having **an internal dataflow which executes all floating point operations using an expanded format that accommodates both floating point formats**. In an embodiment of the present invention, IEEE 754 and IBM S/390 hexadecimal formats are both supported by an expanded hexadecimal format to support IEEE 754 floating point numbers. This ***internal*** **hexadecimal format** supports the full exponent range of both IEEE 754 and S/390 with a 14 bit hexadecimal exponent biased by 8192.

Id. at 3:19-30 (emphasis added). Figure 1 (copied *supra* on page 99), which "provides an overview of the dataflow of Floating Point Unit (FPU), in accordance with the present invention," depicts this ***internal*** floating point format as "HEX-I" and "HEX INTERNAL DATAFLOW." Thus, claim 11 itself, as well as the specification, demonstrates that having an ***internal*** data flow means the FPU uses an ***internal*** floating point format.

IBM relies on the *Microsoft Press Computer Dictionary* (2nd Ed. 1994) for the proposition that "data flow" means "the movement of data through a system from its entry to its destination," and states that its proposed construction "captures this notion." IBM Br. at 62. But

IBM misses the point: even if its proposed construction captures the notion of the extrinsic definition of "data flow," it ignores the intrinsic evidence about what "***internal*** dataflow" means. In light of the intrinsic evidence, PSI's proposed construction is more appropriate.

3. "(a floating point unit that) supports both said first floating point architecture and said second floating point architecture" (claim 11)

IBM's Proposed Construction	PSI's Proposed Construction
Provides the necessary resources for the correct operation of both the first and second floating point architectures.	A floating point unit "supports" a floating point architecture when it can, on its own, perform calculations on floating point numbers of that architecture.

If the Court adopts PSI's construction for "floating point unit," PSI would have no objection to IBM's proposed construction here. Indeed, the Court likely sees no material difference between the two proposed constructions here. That is because there is no material difference—so long as a "floating point unit" is a concrete device and not an amorphous "part of a computer system" that can include software programs.

If the Court does not adopt PSI's proposed construction, then PSI believes its proposed construction for "supports..." is more appropriate because IBM's proposal does not reflect the fact that the purpose of the '106 invention is to avoid the duplication of floating point hardware by providing a single FPU that can, on its own, perform floating point operations in two different floating point architectures:

> There is a need, therefore, to have a computer system that supports two floating architectures. Moreover, such a computer system should provide the two floating point architectures with high performance and with limited resources. That is, multiple floating point architecture support should be provided **without having to essentially duplicate floating point hardware** and without sacrificing performance.

Id. at 1:44-50 (emphasis added). Notably, PSI agrees with IBM that a FPU that "supports" two floating point architectures would have to "provide the necessary resources for the correct operation of both" of those architectures. What is unclear is ***why IBM is unwilling to accept***

PSI's "on its own" limitation: what device, other than the floating point unit, does IBM believe should be construed to provide the necessary resources? The intrinsic evidence indicates that it is the FPU alone that "supports" the two architectures: claim 11 itself, from which the phrase in dispute is taken, refers to "a floating point unit…which supports both said first floating point architecture and said second floating point architecture." *Id.* 22:11-14. No other device is said to "support" the two floating point architectures. Accordingly, the Court should either (1) adopt PSI's construction and/or (2) indicate that there is no difference between these two proposed constructions, since a "floating point unit" is a single device and not an amorphous concept as IBM contends.

4. "converter" (claim 11)

IBM's Proposed Construction	PSI's Proposed Construction
Hardware and/or software that changes data from one format or architecture type to another format or architecture type.	Circuitry within the floating point unit that changes the format or architecture type of a floating point number.

This term presents two issues: (1) whether a "converter" is part of (*i.e.*, located within) the floating point unit that is claimed in the '106 patent and (2) whether a converter is circuitry. As an initial matter, PSI notes that IBM's proposed construction is yet another example of functional claiming: "hardware and/or software" can, in the context of a computer system, mean literally anything. Indeed, IBM's proposed construction, "hardware and/or software that changes data from one format….to another format," would mean that a "converter" includes printers (which convert digital data to printed data), scanners (which convert printed data to digital data), monitors (digital data to visual data), CD players (digital data to audio data), Russian-to-English translation software, and a host of other examples of "hardware and/or software" that it is simply absurd to suggest should be included in the construction of "converter" as that term is used in the '106 patent. IBM's construction should be rejected on its face.

The intrinsic evidence leaves no doubt that a "converter" is necessarily ***part of*** the floating point unit. Every single one of the five claims of the '106 patent asserted in this action notes that what is claimed is "a floating point unit that supports a first floating point architecture and a second floating point architecture." *Id.* at 22:7-9,35,42,53,62. In a section entitled "Overview of Hardware Support for Multiple Architectures," the specification states the following:

> The actual hardware ***needed*** **to** ***support*** **both architectures** consists of ***4 types of format conversion hardware***, a rounder, sticky bit determination, and controls for special number handling.

Id. at 3:49-52 (emphasis added). Thus, in order to support the multiple floating point architectures, the floating point unit "needs" to have certain "actual hardware," including 4 ***converters***. Furthermore, this "hardware" is depicted in Figure 1 (the converters are shaded in red, the other hardware devices in blue):

If a "converter" is not a specific type of structure (*i.e.*, circuitry within the FPU), then claim 11 fails to recite any structure and should be construed to be a means-plus-function claim. *Massachusetts Institute of Technology v. Abacus Software*, 462 F.3d 1344, 1353 (Fed Cir. 2006) ("The presumption that a limitation lacking the term 'means' is not subject to section 112 ¶ 6 can be overcome if it is demonstrated that the claim term fails to recite sufficiently definite structure.") (internal citations omitted). In that case, PSI's construction would *still* be correct because the only structure linked to the four "converter" means is the circuitry within the FPU depicted in Figure 1. **The prosecution history makes precisely this point**. After initially rejecting the claims as obvious in light of several prior art references, the examiner allowed the claims and stated the following in the notice of allowance:

> The prior art of record does not fairly suggest in a computer system a conversion means as recited in claims 1, and **the first through forth [*sic.*] converter[s] as recited in claim 11 wherein, in accordance with 35 USC 112, 6th paragraph, the conversion means is construed to cover the corresponding structure of the combination of converters 11, 12, 15, 16, 22, 24, multiplexers 28, 28, 29 and register 13, 14, 14 connected as disclosed in figure 1** or equivalents thereof, and the first and forth [*sic.*] converter apply the same transformation for both converting an operand of a first floating point architecture type to the internal floating point format, and transforming an operand of a second floating point architecture type as recited in claim 11.

Notice of Allowance dated 4/11/1997 at ¶ 1 (Ex. 33). Thus, if IBM's non-specific "hardware and/or software" construction is to be taken seriously, then claim 11 should be construed as a means-plus-function claim (since a "converter" does not constitute any specific structure), and the corresponding structure for the four converters in claim 11 should be held to be the combination of "converters," "multiplexers," and "registers" depicted in Figure 1. As discussed above, the specification leaves no doubt that these various devices in Figure 1 (a) are hardware and (b) are part of the floating point unit, since the floating point unit "needs" to have them in order to "support" both floating point architectures and since it is undisputed that Figure 1 depicts a floating point unit with this hardware *inside* it. '106 patent at 3:46-52.

It is also worth noting that in the prosecution history the '106 inventors defined "floating point operation" to include "conversion" when they distinguished their invention over the prior art. *See* Applicant's Remarks dated 12/11/1996 at 16 (Ex. 28). The parties agree that, in the context of the '106 patent, floating point units necessarily "perform floating point operations." *See* IBM Br. at 60. PSI respectfully submits that a floating point unit could not ***perform a conversion*** (*i.e.*, an example of a floating point operation) unless that floating point unit included a ***converter***.

With respect to whether a "converter" is circuitry, IBM acknowledges that the patent discloses hardware converters but argues that "[t]he '106 specification also describes conversions 'performed by a *software routine*' for extended operands." IBM Br. at 64, citing 7:57-59. That argument lacks merit and, worse, is misleading. The floating point unit claimed in the '106 patent "supports" multiple floating point architectures by converting operands from these architectures into an internal format, performing the desired floating point calculations in this internal format, and then converting the output of the calculation back to the original architected format. *Id.* at 3:20-24, 22:6-34. The specification tells us that "[t]he actual hardware needed to support both architectures consists of 4 types of format conversion hardware," col. 3:49-50, and Figure 1 makes clear why 4 different types of converters are needed:

1. To perform floating point calculations on operands that are in a hexadecimal architecture ("HEX-A") format, the FPU converts from HEX-A to the internal format ("HEX-I"), performs the operation, then converts back. This requires two types of converters: (1) a "HEX-A TO HEX-I" converter (depicted as 11 and 12) and (2) a "HEX-I TO HEX-A" converter (depicted as 24).

2. To perform floating point calculations on operands that are in a binary architecture ("BIN-A") format, the FPU needs two additional converters: a "BIN-A TO HEX-I" converter (depicted as 15 and 16) and a "HEX-I TO BIN-A" converter (depicted as 21).

Within each *architecture* (*i.e.*, HEX-A or BIN-A), operands may be expressed in different *formats*, including "short," "long," or "extended." *See id.* at 2:40-50. The short format

is the least precise and requires the fewest bits; the extended format is the most precise and requires the most bits. In the same section of the specification from which IBM quotes the reference to a "software routine," the patent tells us that the internal floating point format of the FPU (*i.e.*, HEX-I) has only 64 fraction bits. *Id.* at 7:19-23. That is not enough bits to accommodate the extended HEX-A and BIN-A formats, which have 112 and 113 fraction bits, respectively[30]—in other words, the FPU simply does not have sufficient bits to accommodate these two extended formats. Instead, the specification tells us that when extended-format operands are encountered "the data is partitioned into 3 groups." *Id.* at 7:38-39. Finally, we see the statement IBM selectively quotes from: "For the extended operands, the ***operations*** are performed by a software routine which operates on the 3 portions of data." *Id.* at 7:57-59 (emphasis added).

The reason IBM's brief is misleading on this point is because IBM says that "conversions" are performed by a software routine, whereas the patent actually says "operations" are performed by a software routine:

> IBM Brief at 64: "The '106 specification also describes conversions 'performed by a *software routine*' for extended operations. *Id.*, col. 7:57-59, emphasis added." (Underline added.)
>
> '106 patent at 7:57-59: "For the extended operands, the operations are performed by a software routine which operates on the 3 portions of data." (Underline added.)

The "operations," of course, are not "conversions" at all: they are the actual floating point operations (*e.g.*, add, multiply, etc.) that the user wants to perform on the operands. The FPU in the patent would ordinarily perform these operations ***after converting*** the operands into its internal HEX-I format, but in the case of extended operands the FPU ***cannot perform the conversion*** because it lacks sufficient fraction bits (*i.e.*, it only has 64, not the 112 and 113

[30] *See* Table 1 at col. 2:40-45 (stating that the HEX-A extended format has 112 bits and the BIN-A format has 113 bits).

fraction bits that the extended formats require). Accordingly, the ***operation*** has to be performed by a software routine, a routine which the patent neither describes nor claims. Because this undisclosed software routine clearly does not perform ***conversions***, the examiner did not include it in his recitation of corresponding structure linked to the "conversion means" in claim 1 or the four "converters" in claim 11. *See* discussion *supra* at pages 106-107. In sum, IBM's attempt to use the specification's single mention of a "software routine" to support its construction lacks merit.

IBM's brief also makes the following statement:

> The '106 inventors incorporated flexibility into the specification by not limiting the claims to the disclosed embodiments and expressly stating that a person of ordinary skill in the art would understand equivalent implementations, such as using hardware, **software**, or a combination of the two. *See Id.*, col. 20:47-56.

IBM Br. at 64-65 (emphasis added). That statement is also misleading: the cited portion of the specification (20:47-56) does not mention "software" ***at all***. Instead, that passage in the specification is boilerplate language that appears in all ten patents-in-suit[31] and is irrelevant to the proper construction of "converter."

IBM's final argument is that the format of claim 11 indicates that the converters ***must*** be separate from the FPU:

> Independent claim 11 of the '106 patent recites the "floating point unit" and each of the four "converters" as separate elements, with no language limiting the location of any of the converters within the floating point unit. Because the "converters" are recited as separate claim elements from the "floating point unit," the "converter" ***cannot be*** "within the floating point unit" as PSI proposes.

IBM Br. at 65 (emphasis added). Not surprisingly, **IBM does not cite a single authority to support this claim construction argument**—nor can it, since the argument is obviously a non sequitur. In the prosecution history, the examiner identified the converters in Figure 1 (depicted

[31] *See* '106 patent at 20:47-56; '678 patent at 4:54-58; '709 patent at 4:38-48; '789 patent at 15:11-17; '495 patent at 11:9-19; '261 patent at 19:14-20; '520 patent at 16:1-9; '002 patent at 8:19-30; '812 patent at 10:66-11:4; and '851 patent at 31:65-32:2.

as 11, 12, 15, 16, 21, and 24) as corresponding structure for all of the "conversion" means in the patent, including the four converters in claim 11. *See* discussion *supra* at pages 106-107. It is ***undisputed*** that what Figure 1 depicts is a floating point unit with this set of converters located ***within*** it. *See* IBM Br. at 65, acknowledging that "the '106 specification includes a set of converters within the floating point unit (*see* '106 patent, Fig. 1)." Thus, the intrinsic evidence flatly refutes IBM's argument.

Furthermore, there is absolutely no disclosure in the '106 specification of any floating point unit that does not have converters in it and that, instead, has to somehow utilize external converters. If IBM's "the converter ***cannot*** be within the floating point unit" construction of claim 11 is correct, then claim 11 is invalid: it could not possibly have any corresponding structure, nor could it possibly be enabled.

B. The '678 "Data Class" Patent

The IEEE 754-1985 standard teaches that there are 12 possible data classes for floating-point numbers, depending on their sign, value and exponent. One way to indicate which class a floating-point number is in is to set a "condition code" in the PSW. At the time of the invention of the '678 patent, the condition code in the PSW of IBM System 390 mainframes was limited to 2 bits.[32] It is not possible to represent 12 distinct classes with only 2 bits (2 x 2 = 4)—you need a minimum of 4 bits (2 x 2 x 2 x 2 = 16) to do so. It is this specific problem that the invention in the '678 patent solves:

> Although previous hardware implementations of floating-point arithmetic have provided various radixes, including binary, decimal, or hexadecimal, only a single radix was supported in any particular implementation. As future machines are built, however, they must be compatible with previous machines and must also provide support for new formats. Thus, a new requirement emerges to provide ***hardware support*** for more than one format. In particular, there is a requirement

[32] This hardware limitation was specific to IBM and was not a general problem. For example, the Intel 8087 processor (dating back to 1984) had 4 bits in its condition code (Ex. 37).

> to support both the IBM System/360 hexadecimal and the IEEE binary floating-point formats. This results in several ***unique problems*** which must be solved.
>
> Current instructions, such as Load And Test, which test the state of a floating point number, set the condition code to indicate the sign and value of the number. With IEEE floating-point numbers, there are 12 possible combinations of value and sign. ***This number of combinations, however, cannot be accommodated in the condition code of the program status word (PSW).***
>
> Accordingly, there is a need in the art for an operation that will provide more complete information on the data class of a floating point number.

'678 patent at 1:29-51 (emphasis added). Notably, the Intel processors in PSI's machines have 4 bits in their condition codes, so that they *can* accommodate the 12 floating-point data classes and have absolutely no need for the IBM-specific invention in the '678 patent.

The parties dispute the construction of four claim terms in the '678 patent. Two of the terms, "processor" and "program status word," are discussed above in Section III. The two other terms in dispute are "floating point processor" and "a machine instruction." The parties also dispute the corresponding structure for the means-plus-function elements of claim 1.

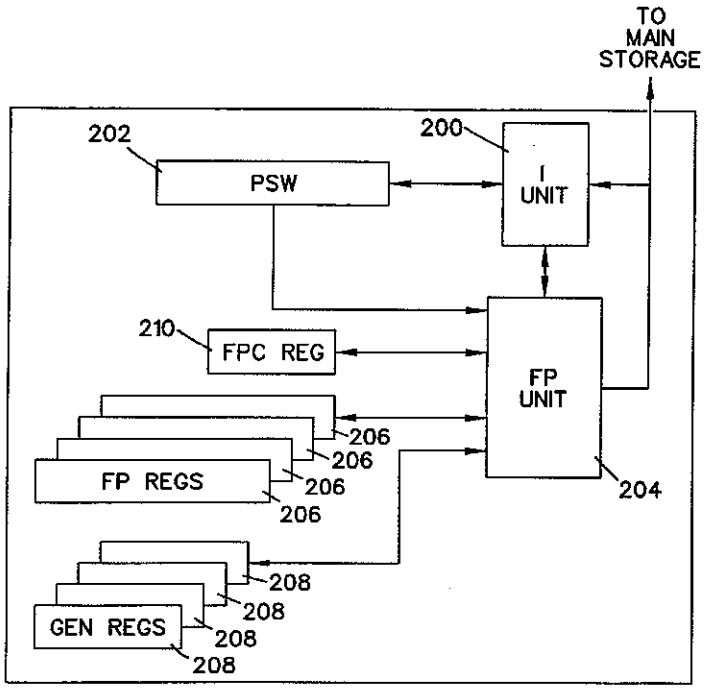
1. "a floating point processor" (claim 1)

IBM's Proposed Construction	PSI's Proposed Construction
A portion of a computer system that interprets and executes floating point instructions.	A processor that contains a floating point unit. A floating point unit is that portion of a processor's circuitry that executes floating point instructions by performing operations on floating point numbers.

The parties' dispute concerning "floating point processor" presents the same issue as the dispute over "processor": should "floating point processor" be defined by its function alone, so that any "portion of a computer system" can be considered to be a "floating point processor" (IBM's position), or does the term refer to the processor circuitry in the intrinsic evidence whose "hardware" limitations the '678 patent was expressly intended to overcome (PSI's position)?

The '678 patent does not expressly define "floating point processor." The term occurs only twice anywhere in the intrinsic evidence, at 3:52 and at 4:63. Nevertheless, it is clear from the patent, and the parties agree, that a floating point processor is something that "executes floating point instructions." IBM's proposed construction does not say what that something that executes is. Instead, IBM's construction is an example of functional claiming: a floating point processor is whatever performs the function of interpreting and executing floating point instructions. PSI's construction posits that a floating point processor is a processor that contains a floating point unit ("FPU").

The '678 specification clearly teaches that floating point instructions are executed by FPUs. For example, the specification illustrates a floating point processor in Figure 2:



'678 patent at Fig. 2. The specification then states the following:

> Instruction unit 200 appropriately hands off retrieved **floating point instructions to floating point unit** 204, along with some of the operands that may be required **by the floating point unit to execute the instruction**. **Floating point (FP) unit 204 includes all necessary hardware to execute the floating point instruction set**, and preferably, in accordance with an embodiment of the present invention,

> supports both Binary and Hexadecimal floating point formats. FP unit 204 is coupled to floating point (FP) registers 206, which contain floating point operands and results associated with **FP unit 204 processing**, and is also coupled to general registers 208. FP unit 204 is also coupled to floating point control (FPC) register 210, which preferably includes mask bits in addition to those provided in the PSW. In a multi-user application, FPC register 210 is under control of the problem state program.

Id. at 2:40-59. Thus, the patent states in direct terms that FPUs are what "execute" floating point instructions. Furthermore, the patent speaks of "FP unit processing," and it notes that the FPU has associated "registers," which are characteristic of processors. The only characteristic that is described as being "preferable" is that the FPU "supports both Binary and Hexadecimal floating point formats." Nothing in the intrinsic evidence, however, even remotely suggests that something other than a FPU executes floating point instructions.

Notably, "floating point unit" is a term used throughout the '106 patent, whose application is incorporated into the '678 specification by reference. *See* '678 patent at 1:22-26. As discussed in Section VI.A.1, IBM's proposed construction for "floating point unit" is essentially identical to its proposed construction for "floating point processor":

IBM's Construction of "floating point processor"	**IBM's Construction of "floating point unit"**
A portion of a computer system that interprets and executes floating point instructions.	Part of a computer system that performs floating point operations.

With that in mind, IBM's own brief illustrates the absurdity of allowing functional claiming:

> A floating point processor can include a floating point unit, but…[t]he '678 specification and prosecution history do not state that a floating processor *must* include a floating point unit, thus leaving open the possibility that a floating point processor can interpret and execute floating point instructions in other embodiments *that do **not** include a floating point unit*.

IBM Br. at 54 (emphasis added). But given IBM's unbounded constructions that argument is silly: how could "a portion of a computer system that interprets and executes floating point

instructions" ***not*** include "a part of a computer system that performs floating point operations"? IBM's constructions defy both the intrinsic evidence and common sense.

The extrinsic evidence also supports PSI's proposed construction. For example, IBM relies on the following definition of "processor" in its brief:

> In a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic and logic unit.

IBM Dictionary of Computing at 533. Consistent with that definition, under PSI's proposed construction of a *floating point* processor is a processor that, in addition to an instruction control unit and an arithmetic and logic unit, also contains a *floating point* unit. Other relevant instances of extrinsic evidence that support PSI's construction include the following:

1. *See* S. Dao-Trong & K. Helwig, "A Single-Chip IBM System/390 Floating-Point Processor In CMOS," *IBM Journal of Research & Development*, Vol. 36, Issue 4, pp. 733-49 (1992) (Ex. 34), which makes clear that "floating point processors" are hardware devices that include circuitry that performs the floating point operations and which states, *inter alia*, the following: "Floating-point processors used to be seen as an option which could be added to the main CPU for performing scientific applications. In the models of the entry-level type (designated 9221) of the new IBM Enterprise System/9000 (ES/9000) line, floating point processing is becoming an inherent part of the CPU. This paper describes the IBM ES/9000 Type 9221 floating-point processor, which is tightly coupled to the CPU and carries out all IBM System/390 floating-point instructions....All floating-point instructions are hard-wired....All circuits are realized in standard cells except for a floating-point register and a multiplier array macro." *Id.* at 733.
2. *See* E. M. Schwarz, L. Sigal, and T. J. McPherson, "CMOS floating-point unit for the S/390 Parallel Enterprise Server G4," *IBM Journal of Research and Development*, Vol. 41,

No. 4/5, pp. 475-488 (1997) (Ex. 32), which discusses the "S/390 floating point unit (FPU)," describes the FPU's various "circuits," and notes that the FPU is "5.3 by 4.7 mm in size." One of the authors of this article, E.M. Schwarz, is a named inventor of U.S. Pat. No. 6,009,261 and U.S. Pat. No. 5,687,106, two of the patents-in-suit.

3. *See* U.S. Pat. No. 7,117,389 (filed Sept. 18, 2003) (Ex. 5), assigned to IBM, which has a "Background" section discussing "the information revolution" and in particular the "central processing unit (CPU) and supporting hardware," and noting the following: "Floating point calculations are among the most time-consuming instructions for processors to perform...[M]ost modern high-performance processors have some form of special floating point functional unit for performing floating point calculations. Due to the complexity of floating point calculations, a floating point unit is generally implemented as a pipeline. Such a pipeline typically requires a significant area of an integrated circuit chip, composed primarily of custom logic. Even though floating point operations are not initiated in every cycle, the circuits within a floating point pipeline have a high degree of switching activity, and consume considerable power at the operating frequencies typical of such devices." *Id.* at 1:24&33, 2:35-48.

4. IBM's patent license with Intel defines "processor" as "an Integrated Circuit which may not include storage but does include circuitry for performing arithmetic and/or logic operations and/or program instruction decoding, interpretation and execution." (Ex. 10).

In sum, both the intrinsic and extrinsic evidence support PSI's proposed construction for "floating point processor."

2. "a machine instruction" (claims 1, 3, 7, 9)

IBM's Proposed Construction	PSI's Proposed Construction
A string of digits that specifies an operation and identifies its operands, if any, and can be directly executed *by a*	A string of digits that specifies an operation and identifies its operands, if any, and can be directly executed *by the*

IBM's Proposed Construction	PSI's Proposed Construction
processor of a computer.	*processor to which it is directed.*

The parties agree that a "machine instruction" is (1) a string of digits (2) that specifies an operation and identifies its operands, if any and that (3) can be ***directly*** executed. The parties' only disagreement is *by what* the machine instruction is "directly executed." PSI contends that a machine instruction can only be "directly executed" by a processor to which it is directed (*i.e.*, a processor which has the same ISA as the machine instruction is written for). **Relying solely on extrinsic evidence** (in particular its own expert's report), IBM rejects this limitation and contends that a machine instruction can be "directly executed" by any processor whatsoever. The Court was presented with this same issue in connection with the construction of the term "instruction," discussed above in Section III.B, and although additional discussion is presented below PSI hereby respectfully incorporates all of its arguments from Section III.B, since they apply with equal force here.

IBM's construction is contrary to the intrinsic and extrinsic evidence, and it flies in the face of the plain meaning of "directly executed." The Intel processors in PSI's machines are EPIC-architecture processors (*i.e.*, they use the EPIC ISA), and they cannot execute ***any*** instructions written for IBM's System-z ISA, including in particular the "Test FP Data Class" instruction that is taught in the '678 patent. Indeed, IBM's own brief acknowledges that "Intel-based computers…are incapable of understanding and executing IBM instructions." IBM Br. at 7.

Thus, the "Test FP Data Class" instruction exists in a PSI machine only as ***data***, and it is only by executing dozens of EPIC-architecture instructions that the Intel processor can emulate the System-z "Test FP Data Class" instruction. Under IBM's construction, the System-z "Test FP Data Class" instruction would be a "machine instruction" ***in an Intel-based machine*** even though the Intel processor treats it only as ***data*** and ***cannot*** execute it.

IBM's tortured use of "directly executed" is contrary to the intrinsic evidence. The '678 patent defines its invention in terms of hardware limitations and an instruction that can be directly executed ***by that same hardware*** in order to address those limitations. Specifically, the specification states the following:

> With IEEE floating-point numbers, there are 12 possible combinations of value and sign. This number of combinations, however, cannot be accommodated in the condition code of the program status word (PSW). Accordingly, there is a need in the art for an operation that will provide more complete information on the data class of a floating point number.

'678 patent at 1:44-48. The PSW condition code incapable of accommodating the 12 IEEE floating-point data classes is the 2-bit condition code in the IBM System-390 processor, which the patent depicts as bits 18 & 19 in Figure 3. Other processors, such as the Intel processors in PSI's machines, have 4 condition code bits and thus no need for the '678 invention.[33]

The specification leaves no doubt that the "Test FP Data Class" floating point instruction it teaches is "directly executed" only by the particular processor (*i.e.*, hardware) that shares its ISA:

> Instruction unit 200 appropriately hands off retrieved floating point instructions to floating point unit 204, along with some of the operands that may be required by the floating point unit to execute the instruction. Floating point (FP) unit 204 **includes all necessary hardware to execute the floating point instruction set**...

Id. at 2:44-49 (emphasis added). The extrinsic evidence also supports PSI's construction. The *IBM Dictionary of Computing*, for example, defines "machine instruction" as follows:

> **machine instruction** (1) An instruction that can be directly executed by a processor of a computer. A machine instruction is an element of a machine language. (2) An instruction of a machine language. (3) Synonymous with computer instruction.

IBM argues that because its dictionary says "a processor of a computer," IBM's *any* processor of *any* computer interpretation must be correct. But that argument lacks merit: PSI's interpretation

[33] Two bits can accommodate at most 4 combinations (2x2=4). Four bits can accommodate up to 16 combinations (2x2x2x2=16).

(that a "machine instruction" is directly executable only by the processor to which it is directed, *i.e.*, the processor with the compatible ISA) is equally if not more consistent with the *IBM Dictionary of Computing* definition. Indeed, while IBM criticizes PSI for providing "identical constructions" for the terms "machine instruction" and "instruction," IBM's own dictionary notes that "machine instruction" and "computer instruction" (the latter being the only type of instruction at issue in any of the patents) are "synonymous." And the '678 patent itself uses the terms "machine instruction" and "instruction" interchangeably, referring in the claims to "a ***machine instruction*** to determine whether the data class of a floating point number is an identified data class," and in the specification to the same instruction as "the Test FP Data Class ***instruction***." '678 patent at 1:61, 4:63-64 (emphasis added).

In sum, IBM's proposed construction confuses *instructions* with *data* (two completely distinct concepts in computing) and is based on a fallacious understanding of what "directly executed" means. PSI's construction is supported by both the intrinsic and extrinsic evidence and should be adopted.

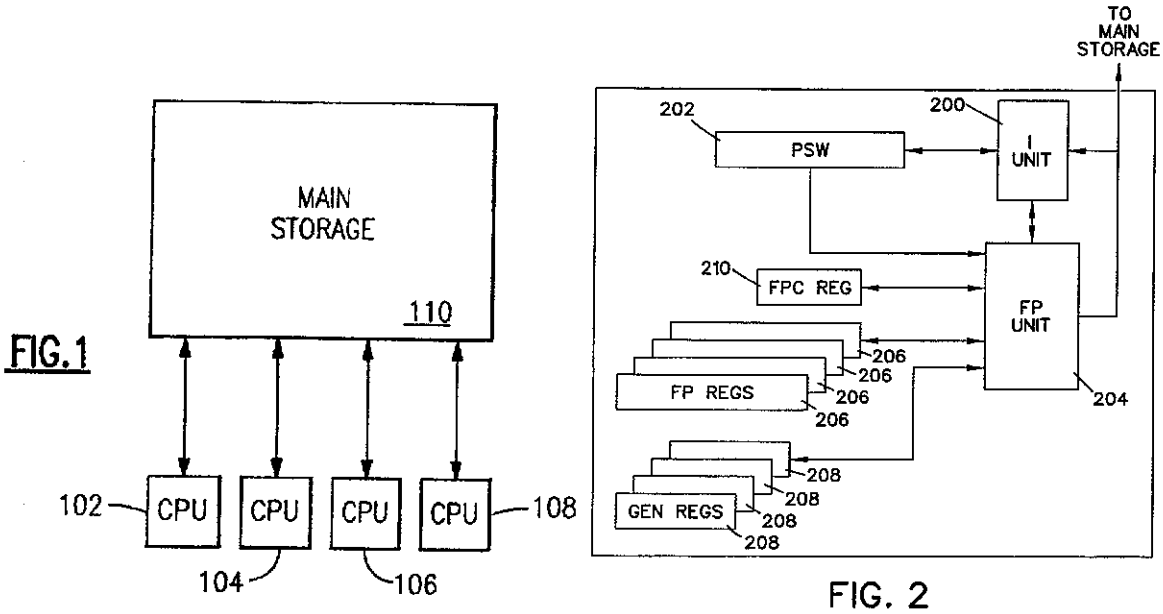
3. "means for retrieving the floating point number from memory" (claim 1)

IBM's Proposed Construction	PSI's Proposed Construction
Function: Retrieving the floating point number from memory. Structure: A shared memory computer system (*see, e.g.*, Fig. 1, 2:35-38; Fig. 2, 2:39-59) for retrieving the floating point number from memory (*see, e.g.*, Fig. 8, 3:63-66), and equivalents thereof.	Function: Retrieving the floating point number from memory. Structure: No corresponding structure.

Whether through neglect or error, the '678 patent simply does not provide any structure that corresponds to the function of "retrieving the float point number from memory" that is recited in claim 1. Instead, the patent does nothing more than (1) refer to a "computer system" (which it depicts with nothing more than boxes labeled "Main Storage" and "CPU") and (2)

provide language which only ***describes*** the function of "retrieving the floating point number from memory." As a matter of law, this is insufficient. *See Aristocrat*, 521 F.3d at 1334 (holding that "language [that] simply describes the function to be performed" is insufficient to constitute corresponding structure under § 112 ¶ 6).

IBM's corresponding structure contentions for the '678 patent amount to pure functional claiming. There are three means-plus-function claim elements in dispute for the '678 patent, with the following three functions: (1) retrieving the floating point number from memory; (2) determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number; and (3) means for setting a condition code in a program status word based upon the determination of whether the data class is the identified data class. For each of these three functions, IBM identifies only a "shared memory computer system" as the corresponding structure, pointing to the same two generic depictions of a general CPU in the specification:



See IBM Br. at 55, 57 & 59 (citing '678 patent at Figs. 1 & 2). For example, for the "means for retrieving the floating point number from memory" limitation, IBM contends that the

corresponding structure is "A shared memory computer system for retrieving the floating point number from memory." IBM Br. at 55. This corresponding structure is facially inadequate under *Aristocrat*. 512 F.3d at 1334 ("In cases involving a computer-implemented invention in which the inventor has invoked means-plus-function claiming, this court has consistently required that the structure disclosed in the specification be more than simply a general purpose computer or microprocessor.").

None of the "examples" in the specification that IBM points to as supporting its "shared memory computer system" contention—col. 2:35-59, 3:63-66 & Fig. 8—constitute adequate corresponding structure, either. The language in column 2, for example, does nothing more than discuss the fetching and execution of ***instructions***, and says nothing about how ***floating point numbers*** are retrieved from memory:

> FIG. 1 illustrates a conventional shared memory computer system including a plurality of central processing units (CPUs) 102-108 all having access to a common main storage 110.
>
> FIG. 2 illustrates functional components included in a CPU from FIG. 1. Instruction unit 200 ***fetches instructions*** [not floating point numbers] from common main storage 110 according to an instruction address located in the program status word (PSW) register 202, and appropriately effects execution of these instructions. Instruction unit 200 appropriately hands off retrieved floating point instructions to floating point unit 204, along with some of the operands that may be required by the floating point unit to execute the instruction. Floating point (FP) unit 204 includes all necessary hardware to execute the floating point instruction set, and preferably, in accordance with an embodiment of the present invention, supports both Binary and Hexadecimal floating point formats. FP unit 204 is coupled to floating point (FP) registers 206, which contain floating point operands and results associated with FP unit 204 processing, and is also coupled to general registers 208. FP unit 204 is also coupled to floating point control (FPC) register 210, which preferably includes mask bits in addition to those provided in the PSW. In a multi-user application, FPC register 210 is under control of the problem state program.

'678 patent at 2:35-59 (emphasis added). Despite its reliance on this passage, even IBM recognizes that this passage says nothing about ***floating point numbers***, and that it only

"discloses the execution of ***instructions*** by the shared memory computer system." IBM Br. at 56 (emphasis added).

IBM next contends that "[t]he '678 specification later discloses (and illustrates in Fig. 8) the remaining algorithm for retrieving the floating point number from memory." *Id.* That disclosure, says IBM, is contained in the following sentence:

> The floating point number in the floating point register that is pointed to by R1 is loaded into the convert to type number logic and a determination is made as to the data class of the number.

Id., quoting col. 3:63-66. There are two problems with IBM's reliance on this passage (and on Figure 8, which IBM says illustrates this passage) as corresponding structure. First, by the time the floating point number is "in the floating point register," ***it has already been retrieved from memory***, since Figure 2 makes clear that the floating point register is located in the CPU. *See* '678 patent at 2:15-16 ("FIG. 2 illustrates functional components included in a CPU from FIG. 1"). Furthermore, this single sentence identified by IBM as the "remaining algorithm" does nothing more than ***describe*** the function: all it does is say that the floating point number is "loaded" and that's it; it does not describe any algorithm for actually carrying out the retrieval from memory. The Federal Circuit's recent opinion in *Aristocrat* is directly on point, and demonstrates that the language identified by IBM is insufficient. *See Aristocrat*, 521 F.3d at 1334 (holding that "language [that] simply describes the function to be performed" is insufficient to constitute corresponding structure under § 112 ¶ 6). Like the patent in *Aristocrat*, the '678 patent simply fails to provide any adequate corresponding structure for this means-plus-function element.

4. "means for determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number" (claim 1)

IBM's Proposed Construction	PSI's Proposed Construction
Function: Determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number. Structure: A shared memory computer system (*see, e.g.*, Fig. 1, 2:35-38; Fig. 2, 2:39-59) for determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number (*see, e.g.*, Fig. 5, 3:34-45; Fig. 7, 3:49-50; Fig. 8, 3:66-4:5; Fig. 9, 4:9-32), and equivalents thereof.	Function: Determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number. Structure: Fig. 9, except for the circuitry that connects AND gates to the box marked CC in Fig. 9 (known in the art as a "wired OR gate").

The only issue that needs to be decided by the Court for this means-plus-function element is whether the corresponding structure (1) is the circuitry depicted in Figure 9 (apart from the "wired OR gate"), which both parties agree constitutes corresponding structure; or (2) whether the corresponding structure also includes Figures 1, 2, 5, 7, 8 and the various passages in the specification identified by IBM, including for example the language at 2:39-59 (*i.e.*, the same passage identified by IBM as corresponding structure for the "retrieving the floating point number from memory" means). The intrinsic evidence establishes that the only corresponding structure is the circuitry depicted in Figure 9, other than the "wired OR gate."

It is undisputed that the "fields" of the floating point number that are "examined" in order to determine what the floating point data class is are the sign, fraction part, and exponent part. '678 patent at 3:66-4:10. The only structure in the patent that is clearly linked to examining these fields is the conversion circuitry of the "convert to type number logic" depicted in Figure 9:

> In the convert to type number logic, these floating point numbers are classified according to the sign, the fraction part, and the exponent part of the floating point number. That is, the floating point format has sign, exponent, and fraction bit. Based on these three bits, the floating point number may be categorized....***FIG. 9 illustrates the conversion circuitry of the convert to type number logic.***

Id. Thus, the "conversion circuitry" depicted in Figure 9 is clearly linked to the function, and Figure 9 (other than the wired OR gate[34]) is properly construed as being corresponding structure. Furthermore, Figure 9 is ***the only structure*** clearly linked to this function by the specification—**IBM does not identify any language in the specification linking any other structure**. Instead, without citing any intrinsic support, IBM inexplicably asserts that (1) the Figure 9 linking language itself, (2) five other figures that relate to other aspects of the invention and (3) the same purported corresponding structure IBM identified for the function of "retrieving the floating point number from memory" are all corresponding structure. IBM Br. at 57-58. Because none of these various passages or figures are clearly linked to the function of examining the floating point number fields to determine data class, IBM's attempt at wholesale inclusion of various and sundry parts of the specification as corresponding structure should be rejected. *See Braun*, 124 F.3d at 1424 ("Structure disclosed in the specification is 'corresponding' structure only if the specification or prosecution history clearly links or associates that structure to the function recited in the claim").

5. "means for setting a condition code in a program status word based upon the determination of whether the data class is the identified data class" (claim 1)

[34] The wired OR gate is discussed *infra* at page 125. In short, the specification establishes that the wired OR gate is structure linked to the function of setting the condition code, and is not used in examining the fields of floating point numbers. '678 patent at 4:5-10.

IBM's Proposed Construction	PSI's Proposed Construction
Function: Setting a condition code in a program status word based upon the determination of whether the data class is the identified data class. Structure: A shared memory computer system (*see, e.g.*, Fig. 1, 2:35-38; Fig. 2, 2:39-59) for setting a condition code in a program status word based upon whether the data class is the identified data class (*see, e.g.*, Fig. 8, 4:5-8; Fig. 9, 4:9-32), and equivalents thereof.	Function: Setting a condition code in a program status word based upon the determination of whether the data class is the identified data class. Structure: The circuitry that connects the AND gates to the box marked CC in Fig. 9 (known in the art as a "wired OR gate").

This means-plus-function element presents essentially the same single issue as the last one: whether the corresponding structure (1) is the circuitry depicted in Figure 9, which both parties agree is corresponding structure; or (2) also includes Figures 1, 2 & 8, as well as the linking language in the specification and the language at 2:35-59 (*i.e.*, the same passage IBM identified as structure for the previous two means).

The intrinsic evidence demonstrates that only the wired OR gate depicted in Figure 9 is appropriately considered corresponding structure. Figure 9 "illustrates the conversion circuitry of the convert to type number logic of the Test FP Data Class operation." '678 patent at 2:29-30. With the wired OR gate highlighted in red, Figure 9 looks as follows:

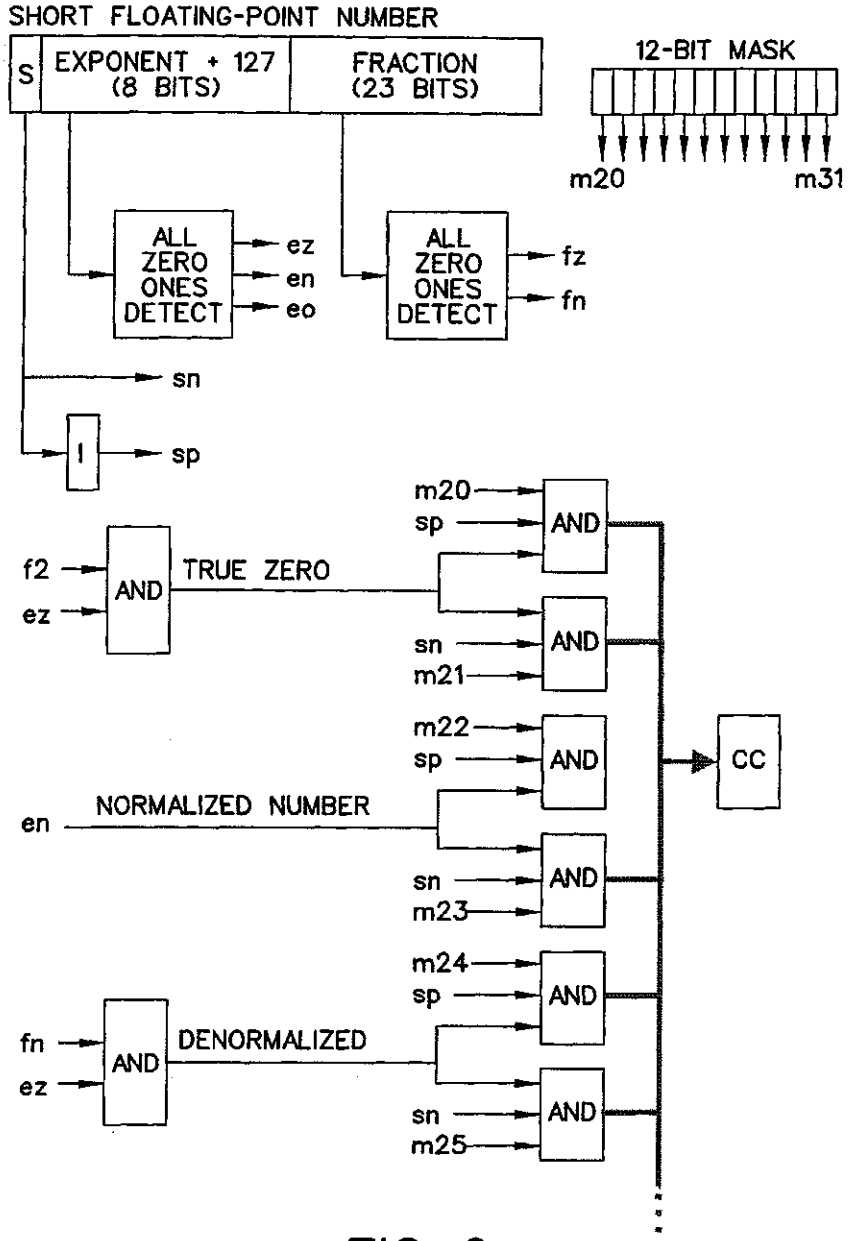


FIG. 9

The circuitry setting the condition code ("CC") is considered an "OR gate" because it follows disjunctive logic in setting the condition code: so long as any one of the AND gates is a "1", the circuit will set the condition code to "1."[35]

[35] For example, if the user wanted to know whether a particular floating point number was true zero, a positive normalized number, or a negative denormalized number (*i.e.*, was in one of any of those three floating point data classes), the OR gate would set the condition code to 1 ("yes") so long as examination of the fields of the floating point number revealed that the floating point number fell into *any one* of those three data classes.

The only structure in the '678 patent that is clearly linked with the "setting a condition code..." function is the wired OR gate depicted above. Specifically, the specification contains the following linking language:

> In the convert to type number logic, these floating point numbers are classified according to the sign, the fraction part, and the exponent part of the floating point number....Based upon the particular data class that it falls into, a signal is generated that will indicate one of the bits in the mask. **That bit is then loaded into the condition code. That is, the condition code is set**.
>
> **FIG. 9 illustrates the conversion circuitry of the convert to type number logic for bits 20 through 25 of the mask for the Test FP Data Class operation**....By way of example, if it was to be determined whether the number is positive zero, bit location 20 in the mask field would be set to 1. Then, if the floating point number were a positive zero the convert to type number logic would cause the mask bit 1 **to be loaded into the condition code storage location**. If a positive zero is not to be tested, a 0 would be set in bit location 20 in the mask field and the 0 **would be loaded into the condition code storage location**.

3:66-4:32 (emphasis added). Notably, **IBM does not identify any linking language in the patent**—it considers everything, including the above passage, to be corresponding structure. IBM Br. at 60. However, it is black letter Federal Circuit law that "[s]tructure disclosed in the specification is 'corresponding' structure only if the specification or prosecution history clearly links or associates that structure to the function recited in the claim." *Braun*, 124 F.3d at 1424. In its rush to grab onto as much structure as possible, IBM neglected to identify ***any*** linking language for this function at all. Because the only linking language is that quoted above, and because the only structure that language associates with the recited function is the wired OR gate connected to the "condition code storage location" in Figure 9, PSI's position should be adopted by the Court.

C. The '709 "Rounding Modes" Patent

The parties dispute two terms in the asserted claims of the '709 patent, "processor" and "instruction." These terms are discussed *supra* in Sections III.A and III.B.

VII. THE PARTITIONING PATENTS

A. The '812 "Partition Communication" Patent

Mainframes are often divided into a number of "partitions" which operate as independent computers. Like many computers, these partitions often need to communicate with one another over a network, such as the internet. Instead of having each partition connected to the network through a separate network interface, mainframe partitions typically all use the same piece of hardware, called a "host-network interface," to handle their communications with the network.

The '812 patent describes a simple efficiency improvement that can be made when multiple partitions are connected through a single host-network interface. When one partition sends data to the host network interface that is directed at another of the partitions, instead of having the host network interface transmit this data out over the network (which is what the '812 patent tells us the prior art did), the host-network interface disclosed in the '812 patent routes the information directly to the receiving partition. '812 patent at 3:45-61. The host-network interface can do this because it stores a list of the network addresses of all of the partitions to which it is connected. *Id.* at 7:45-62. It consults that list each time it receives a packet of information from the host computer in order to determine whether to route the packet internally or whether to send it out over the network. *Id.* If the packet is destined for another partition on the same system, the host-network interface responds by "forwarding the IP datagram directly from the first partition to the second partition without employing the network." *Id.* at 3:59-61.

In short, the system described in the '812 patent is like a clever postman at an apartment building. When this postman sees an outgoing letter from Apartment A that is addressed to Apartment B, he puts it straight in Apartment B's mailbox, without first carrying it back to the post office and having it routed through the mail-sorting process. Since routing a packet out over

the network is slower than simply routing it inside the network interface, the system and method described by the '812 patent increases the speed of network communication between partitions.

The parties dispute the construction of the claim terms "host network interface" and "saving at said host-network interface." The parties also dispute the existence and identification of corresponding structure for the means-plus-function limitations in claim 11.

1. "host-network interface" (claims 1, 11)

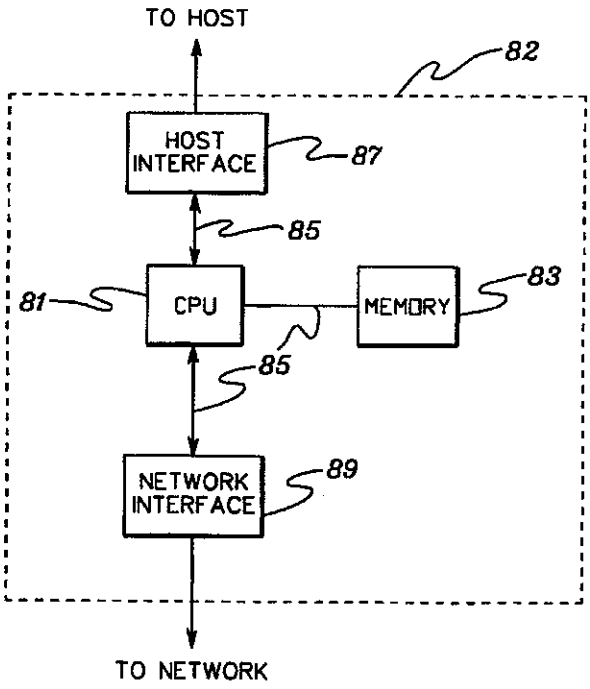
IBM's Proposed Construction	PSI's Proposed Construction
The communication interface between a host computer and a network.	A hardware component coupled between a mainframe class data processing system and a network port.

The parties' constructions present two issues for the Court. First, the parties dispute whether a host-network interface (a) is anything that can be classified as part of a "communication interface" (IBM's position) or (b) is necessarily a "hardware component" of a computer system (PSI's position). Second, the parties disagree about where the host-network interface is. Specifically, the parties disagree about whether it is (a) "coupled between a mainframe class data processing system and a network port" (PSI's position) or (b) need only be somewhere "between a host computer and a network" (IBM's position).

a. "communication interface" vs. "hardware component"

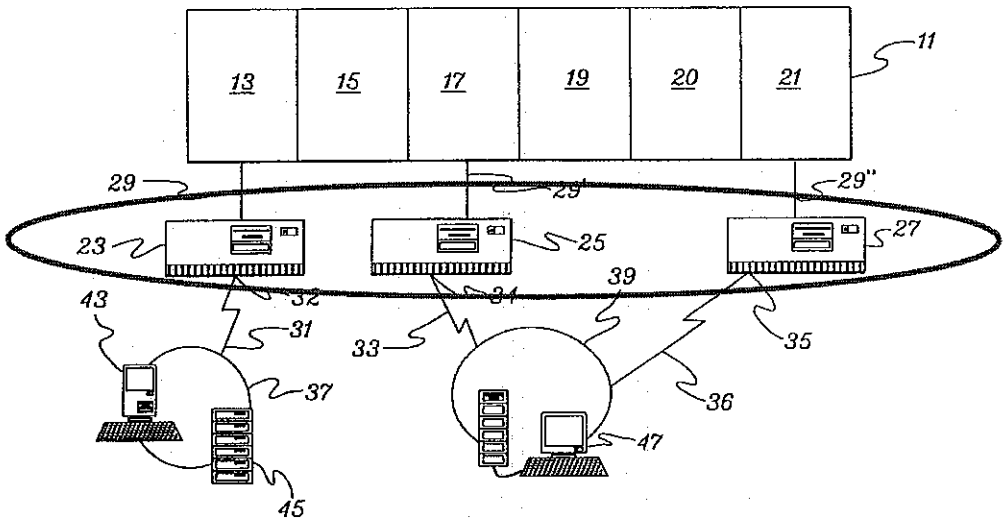
The intrinsic evidence establishes that the "host-network interface" is a specific piece of hardware, and not merely anything that performs the generalized function of a "communication interface." First, the *location* of the host-network interface clearly indicates that it is a piece of hardware. As discussed in more detail below, IBM stated during the prosecution of the '812 patent that the host-network interface "is coupled between the mainframe class data processing system having the multiple partitions and a port to a network." 5/29/2003 Remarks to Amendment (Ex. 35) at 3. When something is coupled between two other physical objects (in this case the mainframe and the port to a network) it must *itself* be a piece of hardware.

Consistent with this statement from the prosecution history, the specification also demonstrates that the host-network interface is a piece of hardware. The specification contains the following general depiction of "a block diagram of a host to network interface":



See U.S. Pat. No. 5,740,438 (Ex. 38) at Fig. 4, incorporated by reference into the '812 specification ('812 patent at 2:40-41). This figure shows that the host-network interface is, as PSI contends, a *hardware* component: it includes a CPU (hardware), memory (hardware), and interfaces with the host and with the network.

The specification also depicts the preferred embodiment IBM model 3172 host-network interfaces (items 23, 25 & 27), which are indisputably hardware devices:



'812 patent at Fig. 1. & 1:66-2:17 (identifying items 23, 25 & 27 as IBM 3172 host-network interfaces). The fact that the patent uses the term "host-network interface" when talking about the hardware IBM model 3172 device supports PSI's construction.

Furthermore, the extrinsic evidence establishes that the term "host-network interface" as used in the claims of the '812 patent refers to the computer's network interface card, or NIC. As IBM states: "The NIC is effectively a method of connecting the internal data bus of a computer to the external media (cables) of the network. In the case of z/OS there is essentially only one NIC: the Open Systems Adapter (OSA) card" (Ex. 36).[36] "The Open Systems Adapter is actually a network controller that you can install in a mainframe I/O cage. The adapter integrates several hardware features and supports many networking transport protocols" (Ex. 37).[37] Because it is a device—a "card"—that can be installed in a particular physical configuration—"in a mainframe I/O cage"—the host-network interface is necessarily, as PSI contends, a *hardware* component.

[36]http://publib.boulder.ibm.com/infocenter/zoslnctr/v1r7/index.jsp?topic=/com.ibm.znetwork.doc/znetwork_24.html.
[37]http://publib.boulder.ibm.com/infocenter/zoslnctr/v1r7/index.jsp?topic=/com.ibm.znetwork.doc/znetwork_57.html.

IBM argues that the host-network interface is not a hardware component because the disclosed embodiment in the patent is hardware that utilizes software, such as the "LAN device drivers" software illustrated in Figure 3. IBM Br. at 67-68. But this argument misses the point: virtually all pieces of hardware in a computer utilize software during their operation. The passage IBM cites, for example, identifies one embodiment of a host-network interface as a "host channel connection 118 and a communications adapter, such as an IBM Open Systems Adapter (OSA) 120." IBM Br. at 68 (citing '812 at 5:12-13). But as noted above, the OSA adapter is a "card"—a piece of hardware. The fact that an OSA adapter card—or a hard drive or an Intel Itanium-2 microchip—utilizes software does not change the fact that the card is itself *hardware.*

Moreover, even if one were to think of the host-network interface as including the software (*i.e.*, being a combination of hardware and software), IBM's proposal (any "hardware and software" that "interfaces" between the computer and the network) is still wrong. Instead, people of ordinary skill in the art would understand that the interface is a hardware *component* that makes use of software stored on a memory *inside the component* to perform its task. It cannot be properly construed as constituting *any* hardware or software involved in sending signals to and from the network because, as a practical matter, this would include everything in the computer system. Instead, the host-network interface is a discrete hardware component that is physically connected to the network port and (through a combination of physical circuits and software stored in a memory within it) performs the tasks needed to send and receive signals coming and going directly from and to the network port. *See* Patt Decl. ¶ 27.

PSI's construction is more appropriate for the additional reason that IBM's construction includes anything that performs the unbounded function of serving as an "interface." The problem with this unbounded construction becomes immediately apparent when one recognizes that it is impossible to know what components are supposed to be inside and what components

are supposed to be outside a "communications interface." Signals and information transmitted over a network can be traced all the way back to the processor or even the disk drive—meaning that, as a practical matter, IBM's construction would include everything in the computer. *See* Patt Decl. at ¶ 27. Under IBM's construction, all of the components in a computer are literally part of the "communications interface" because there is ***nothing*** to say where the "interface" ends and where the computer or the network begins. If IBM's proposal is adopted, it can look at PSI's computer, find ***any*** location where signals are routed between partitions, and claim that that location is part of the "communications interface between the computer and the network." In short, IBM's proposal is unbounded by any physical limitations. Indeed, IBM's construction does not contain any *functional* limitations, either, because everything in a computer "communicates" and all of those communications can, after enough intermediary steps, be sent out over the network. Thus, it is hard to imagine anything in a computer system that does not literally fall within the scope of IBM's proposed construction.

b. "between a host computer and a network" and "coupled between a mainframe class data processing system and a network port"

PSI's construction should be adopted because it is supported by the intrinsic evidence and because IBM's construction is meaningless. During the prosecution history IBM made the following statement to the examiner: "Applicants' technique includes saving at the host-network interface, ***which is coupled between the mainframe class data processing system*** having the multiple partitions ***and a port to a network***, an internet protocol address of at least one of the partitions in the mainframe class data processing system." *See* Remarks to an Amendment filed May 29, 2003 (Ex. 35) at 3. It is black letter law that "[t]he patentee is held to what he declares during the prosecution of his patent." *Gillespie v. Dywidag Sys. Int'l*, 501 F.3d 1285, 1291 (Fed. Cir. 2007). Accordingly, this intrinsic evidence should govern this construction.

Furthermore, IBM's construction regarding the location of the host-network interface is meaningless: the host-network interface, argues IBM, does not need to be connected to the *host* and does not need to be connected to the *network*, but instead can be any component anywhere "between" those two. But as a practical matter ***anything*** inside a computer meets this construction, since everything in a computer is "between" some other part of the computer and the network. This construction not only ignores the intrinsic evidence, it also tells you nothing about the location of the host-network interface.

In sum, IBM's construction effectively asks the court to read all meaning out of the limitation. PSI's construction uses the term as it was used in the patent, in the incorporated reference, in the file history, and in IBM's other published documents. PSI's construction should be adopted.

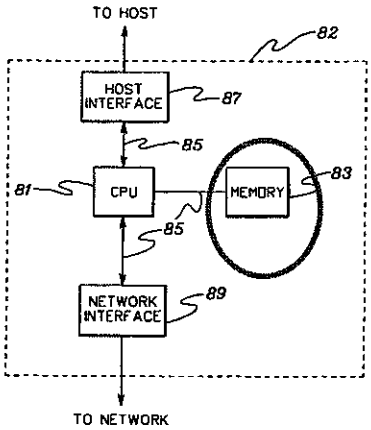
2. "saving at said host-network interface" (claim 1)

IBM's Proposed Construction	PSI's Proposed Construction
Saving at said host-network interface.	Saving on memory located within the host-network interface.

IBM's proposed "construction" of this term is not a construction at all—it simply repeats the language of the disputed claim term. Such attempts to shift the burden of claim construction to the jury by claiming that a contested term should receive its "ordinary meaning" have recently been prohibited by the Federal Circuit. In *O2 Micro Intern. Ltd. v. Beyond Innovation Tech. Co.*, 521 F.3d 1351 (Fed. Cir. 2008), the court held that "[w]hen the parties present a fundamental dispute regarding the scope of a claim term, it is the court's duty to resolve it." *Id.* at 1362. Because there is a dispute here (despite meeting and conferring the parties were unable to agree on a construction), IBM's proposed *non*-construction should be rejected.

PSI's construction should be adopted because the intrinsic evidence establishes that "saving at said host-network interface" means saving ***on memory*** located within the host-

network interface. Specifically, IBM's patents describe the host-network interface as a hardware component that contains its own memory:



See U.S. Pat. No. 5,740,438 (Ex. 38) at Fig. 4, incorporated by reference into the '812 specification ('812 patent at 2:40-41). When data is saved "at" the host-network interface, it is necessarily saved on that memory. There is no other location "at" the host-network interface where the list of network addresses could be saved. Patt Decl. ¶ 28.

Second, the '812 claims explicitly state that the data are saved "in an address resolution protocol (ARP) cache at said host-network interface." '812 patent at 11:23-30 (*i.e.*, claim 2). The ARP cache, in turn, is "contained in the OSA adapter 120," and is represented as item number 123 in Figure 3. *Id.* at 5:52-54. The ARP cache is ***memory*** located within the host-network interface. *Id.* at Fig. 3. Thus, when IP addresses are saved "at" the host-network interface, they are saved "on memory located within the host-network interface." For these reasons, PSI's proposed construction is correct and should be adopted.

3. Means Plus Function Limitations

Claim 11 of the '812 patent contains 4 means-plus-function limitations that are related to one another. Although the parties agree on the *function* for each of these limitations, they disagree on whether there is corresponding structure. Because the issues are essentially the same

for each disputed limitation, they are treated together below, rather than on a limitation by limitation basis.

<table>
<tr><th>Claim Limitation</th><th>IBM's Proposed Construction</th><th>PSI's Proposed Construction</th></tr>
<tr><td rowspan="2">(i) "at least one computer usable medium having computer readable program code means embodied therein for causing network communications in a mainframe class data processing system having multiple partitions and a port to a network, the computer readable program code means in the article of manufacture comprising"</td><td colspan="2">Agreed Function: Causing network communications in a mainframe class data processing system having multiple partitions and a port to a network.</td></tr>
<tr><td>Structure: An article of manufacture (see, e.g., 10:46-57 and equivalents thereof) that causes a computer to cause network communications in a mainframe class data processing system having multiple partitions and a port to a network (see, e.g., Figs. 5, 6, 7, 8A, 8B; <u>or</u> col. 7:45-8:19; 8:31-48; 9:8-10; 9:46-57; 10:1-11), and equivalents thereof.</td><td>Structure: No structure.</td></tr>
<tr><td rowspan="2">(ii) "computer readable program code means for causing a computer to effect saving at a host-network interface an internet protocol (IP) address of at least one of the multiple partitions of the mainframe class data processing system"</td><td colspan="2">Agreed Function: Causing a computer to effect saving at ghost-network interface an internet protocol (IP) address of at least one of the multiple partitions of the mainframe class data processing system.</td></tr>
<tr><td>Structure: An article of manufacture (see, e.g., 10:46-57) that causes a computer to effect saving at a host-network interface an internet protocol (IP) address of at least one of the multiple partitions of the mainframe class data processing system (see, e.g., Fig. 5, 7:66-8:4), and equivalents thereof.</td><td>Structure: No structure.</td></tr>
<tr><td>(iii) "computer readable program code means for causing a computer to effect generating an IP datagram at a first partition of said multiple partitions to be forwarded to a second partition of said multiple partitions using a destination IP</td><td colspan="2">Agreed Function: Causing a computer to effect generating an IP datagram at a first partition of said multiple partitions to be forwarded to a second partition of said multiple partitions using a destination IP address.</td></tr>
</table>

address"	Structure: An ***article of manufacture*** (***see, e.g.***, 10:46-57) that causes a computer to effect generating an IP datagram at a first partition of said multiple partitions to be forwarded to a second partition of said multiple partitions using a destination IP address (***see, e.g.***, step 200 of Fig. 6, 8:33-37; step 300 of Fig. 7, 9:8-10; Fig. 8A, 9:46-57; **or** Fig. 8B, 10:1-11), and equivalents thereof.	Structure: No structure.
(iv) "computer readable program code means for causing a computer to effecting determining whether said destination IP address for said IP datagram comprises an IP address saved at said host-network interface for said at least one partition, and if so, forwarding the IP datagram directly from said first partition to said second partition of said multiple partitions without employing said network"	Agreed Function: Causing a computer to effect determining whether the destination IP address far the IP datagram comprises an IP address saved at the host-network interface for said at least one partition, and if so, forwarding the IP datagram directly from the first partition to the second partition of said multiple partitions without employing the network.	
	Structure: An ***article of manufacture*** (***see, e.g.***, 10:46-57) that causes a computer to effect determining whether the destination IP address for the IP datagram comprises an IP address saved at the host network interface for said at least one partition, and if so, forwarding the IP datagram directly from the first partition to the second partition of said multiple partitions without employing the network (***see, e.g.***, Fig. 5, 8:20-30; **or** steps 210-240 of Fig. 6, 8:37-48), and equivalents thereof.	Structure: No structure.

Once again IBM's proposed structures are facially inadequate. IBM attempts to claim ***all*** "articles of manufacture" that can cause a general purpose computer to perform the function recited in the limitation. This type of unbounded claiming contravenes § 112 ¶ 6 and is improper. *Aristocrat*, 521 F.3d at 1334 ("The point of the requirement that the patentee disclose particular structure in the specification and that the scope of the patent claims be limited to that structure and its equivalents is to avoid pure functional claiming.").

Moreover, the "examples" that IBM cites for each of the claim limitations are generic boilerplate passages that tell the reader nothing about "the corresponding structure, material, or acts" that perform the claimed function, as required by § 112 ¶ 6. The language in the specification that IBM cites as describing the "corresponding structure" for ***all*** of the limitations is the following:

> The present invention can be included, for example, in an article of manufacture (*e.g.*, one or more computer program products) having, for instance, computer usable media. This media has embodied therein, for instance, computer readable program code means for providing and facilitating the capabilities of the present invention. The articles of manufacture can be included as part of the computer system or sold separately. Additionally, at least one program storage device readable by machine, tangibly embodying at least one program of instructions executable by the machine, to perform the capabilities of the present invention, can be provided.

'812 at 10:46-57. But this passage cannot constitute corresponding structure because it does not describe any specific structure for carrying out the claimed function. Instead, the generic language in this paragraph is so general that it describes ***all*** computer programs.

Indeed, this *identical* language appears in *nearly a hundred* unrelated IBM patents.[38] *See, e.g.*, U.S. Patent No. 6,040,861 (Ex. 39) at 10:56-63 (a video encoding patent); U.S. Patent No. 5,978,916 (Ex. 40) at 4:66-5:6 (a patent on digital rights management software); and U.S. Patent No. 6,097,757 (Ex. 44) at 10:50-67 (a patent on a video compression technique). Even in this lawsuit IBM relies on this verbatim language as purported corresponding structure for the means-plus-function limitations in the '789 patent, a patent that IBM does not even consider to be part of the same group as the '812 patent. IBM Br. at 36-37 & 39 (citing verbatim language as corresponding structure for '789 patent).

Generic language describing general-purpose computer-usable media like CD-ROMs and hard drives does not provide sufficient corresponding structure to satisfy the statutory *quid pro*

[38] http://www.google.com/patents?q=%22The+present+invention+can+be+included,+for+example,+in+an+article+of+manufacture%22&lr=&start=0&scoring=2 (providing links to more than 90 other IBM patents and pending patent applications containing identical language)

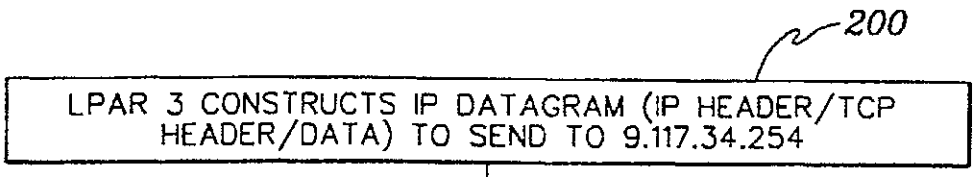
quo of means-plus-function claiming. *Medical Instrumentation*, 344 F.3d at 1211 ("If the specification is not clear as to the structure that the patentee intends to correspond to the claimed function, then the patentee has not paid the price but is attempting to claim in functional terms unbounded by any reference to structure in the specification."); *Biomedino*, 490 F.3d at 948 ("[I]n return for generic claiming ability, the applicant must indicate in the specification what structure constitutes the means."). Accordingly, IBM's corresponding structure contentions should be rejected on their face.

The "examples" that IBM identifies as support for its boilerplate structure are also inadequate in light of *Aristocrat*. That case requires that the specification disclose a "particular algorithm" for achieving the claimed function when that function is implemented in a general purpose computer. *Aristocrat*, 521 F.3d at 1335-36. For example, with respect to limitation (ii) above, IBM cites the following passage as an example supporting its boilerplate structure:

> At initialization time, each Host TCP/IP stack is configured to register its HOME IP addresses with the OSA adapter ***in a manner apparent to one skilled in the art***. The "HOME" addresses are those which are recognized as local IP addresses by the specific stack. These entities are marked as HOME entries in the ARP cache.

'812 patent at 7:66-8:4 (emphasis added). Rather than stating ***how*** to structure a machine or program to achieve the claimed function, the cited passage merely states that the necessary programming would be "apparent to one skilled in the art." But that is precisely the sort of shortcut language that the Federal Circuit rejected in *Aristocrat*. 521 F.3d at 1334 (finding no corresponding structure where the specification merely called for "appropriate programming").

With respect to limitation (iii), the structure is described in even more vague and general terms. Specifically, IBM cites "step 200 of Fig. 6, 8:33-37," which consists of the following figure and language:



> In this example, a logical partition, LPAR 3, is assumed to construct an IP datagram which includes an IP header, a TCP header and data, for sending to an IP address 200, which is assumed to comprise address 9.117.34.254 (see FIGS. 3 & 5).

But no algorithm for "construct[ing] the IP datagram," as the disputed passages require, is provided in the above portions of the specification. For example, no instructions are provided in the cited passage for how the "IP header," "TCP header" and "data" are structured, nor are we told what is contained in the "IP header" or the "TCP header." Further, by using language like "this example," "is assumed to construct," and "is assumed to comprise," the cited passage does not state with specificity what the relevant structure actually ***is***. A person of skill in the art, upon reading the passage, is left to guess whether the construction of the IP header is an essential part of this claim limitation, or whether it is merely part of the "example."

Similarly, with respect to limitation (iv) above, IBM cites the following passage:

> The HOME entries serve another purpose. In accordance with a further aspect of the invention, these entries are preferably used to route packets from a first logical partition (LPAR) to a second LPAR of the host system without going onto a network. When a packet is received from the Host, the destination IP address is "looked up" in the ARP tables of the host-network interface. If an entry is found and it is marked as a HOME entry, then the IP packet is routed directly to the LPAR owning that address. Since the packet is not sent out onto a network, no media or network header needs to be constructed.[39]

'812 patent at 8:20-30. This passage is not much more specific than the "function" language itself, which calls for "determining whether the destination IP address for the IP datagram comprises an IP address saved at the host-network interface for said at least one partition." It adds only that the relevant "determining" can be performed by "look[ing] up" the destination IP

[39] '812 patent at 8:20-30.

address "in the ARP tables of the host-network interface." But it does not, as *Aristocrat* requires, provide an algorithm that actually enables one to practice the recited function.

The above discussion demonstrated that the "examples" IBM points to for limitations (ii), (iii) and (iv) are inadequate. The same analysis applies to limitation (i), since the "examples" IBM points to for that limitation are merely a compilation of the same "examples" IBM cites for limitations (ii) through (iv).

IBM has failed to identify valid corresponding structure for these claim limitations. PSI's construction should be adopted, and, as in *Aristocrat*, the claim should be held invalid.

B. The '002 "Firmware Booting" Patent

At a high level, firmware is software that is stored in a location that cannot be accessed by the user or overwritten by the operating system. Because it is protected against erasure, firmware is used by system designers to handle critical tasks, such as starting up or "booting" the computer. As noted previously, mainframes are divided into partitions that operate largely as if they were independent computers. When a partition is turned on, it runs firmware during the booting process in order to prepare the hardware to start the operating system.

It is inefficient in a mainframe to restart the entire system if only a single partition crashes and needs to be rebooted. For that reason, mainframe designers have developed ways of restarting one partition without restarting the others. The '002 patent claims a system and method of rebooting a single partition using firmware. In particular, the '002 patent claims a system in which a given partition can be restarted using either of two *different* firmwares. This means that, when you restart the partition, you can *change* its configuration: you can restart it with one of the firmwares if you want it to function one way, or with the other firmware if you want it to function a second way.

The parties dispute the construction of the claim terms "firmware," "firmware image," and "capable of being executed during a power-on process to boot said computer system." The parties also dispute whether the specification identifies corresponding structure for two means-plus-function limitations in claim 9.

1. "firmware" (claims 1, 9, 17)

IBM's Proposed Construction	PSI's Proposed Construction
(IBM does not propose a construction.)	"Software" stored in a memory chip that holds its content without electrical power, such as, for example, read-only memory (ROM), programmable ROM (PROM), erasable programmable ROM (EPROM), electrically erasable programmable ROM (EEPROM), and non-volatile random access memory (non-volatile RAM).

The specification expressly defines the term "firmware." '002 patent at 6:22-24. Nevertheless, IBM refuses to propose a construction for the term. This gambit to prevent the Court from construing the term should fail under *O2 Micro Intern. Ltd. v. Beyond Innovation Tech. Co.*, 521 F.3d 1351 (Fed. Cir. 2008) ("When the parties present a fundamental dispute regarding the scope of a claim term, it is the court's duty to resolve it."). The scope of the term firmware is, in fact, disputed: IBM disagrees with PSI's contention that, in the context of the '002 patent, "firmware" is software that is "stored ***in a memory chip*** that holds its content without electrical power." Instead, IBM contends that firmware is software in ***any*** non-volatile memory. IBM Brief 79-80. Thus, there is a live dispute over the scope of the term "firmware" that the Court should resolve.

IBM wants to avoid this dispute because the intrinsic evidence overwhelmingly supports PSI's construction. PSI's construction is mandated by the fact that the patentee acted as a lexicographer with respect to this term. '002 patent at 6:22-24. As the Federal Circuit has unequivocally stated, "the specification 'acts as a dictionary when it expressly defines terms used

in the claims or when it defines terms by implication.'" *Phillips*, 415 F.3d at 1321. "[W]here the patentee, acting as his or her own lexicographer, has clearly set forth an explicit definition of the term," that definition controls. *Id.* at 1319.

The specification defines "firmware" as follows:

> Firmware is "software" stored in a memory chip that holds its content without electrical power, such as, for example, read-only memory (ROM), programmable ROM (PROM), erasable programmable ROM (EPROM), electrically erasable programmable ROM (EEPROM), and non-volatile random access memory (non-volatile RAM).

'002 patent at 6:22-24. The specification's definition of "firmware" is, verbatim, the definition PSI advocates here. The patentee has acted as his own lexicographer, and IBM cannot escape the definition of "firmware" it set forth in the patent specification—especially by refusing to propose a construction and insisting instead that the term need not be construed.

Although dictionaries are not necessary where, as here, the patent expressly defines a term, it is worth noting that the specification's definition of "firmware" is similar to extrinsic dictionary definitions. For example, the *Compact Oxford English Dictionary* defines "firmware" as "permanent software programmed into a read-only memory" (Ex. 42). The *Microsoft Computer Dictionary* defines "firmware" as "Software routines stored in read-only memory (ROM)" (Ex. 30). And the *McGraw-Hill Dictionary of Scientific and Technical Terms* defines "firmware" as "[a] computer program or instruction, such as a microprogram, used so often that it is stores in a read-only memory instead of being included in software" (Ex. 43).

Furthermore, as the variety of dictionary definitions suggests, the term "firmware" has no singular fixed meaning to people of ordinary skill in the art. Patt Decl. ¶ 29. "Firmware" was closely associated with "microcode" in the 1960s. But by the time the patents in suit were filed in the 1990s, people regularly used and understood the term "firmware" to refer to code stored in all kinds of different media with different degrees of protection from erasure and different

relationships to the various other architectural layers in a computer system. *Id.* As a result, the word "firmware" – in isolation – had no specific meaning to a person of ordinary skill in the art at the time of the inventions of the patents in suit. And such a person would have had to deduce the meaning of the term "firmware" from whatever meaning it was given in the patent. *Id.* 30-31.

Notably, in other legal proceedings between IBM and PSI, IBM itself has represented that the term "firmware" refers to software "burnt" on a memory chip that holds its content without electrical power. Specifically, in [redacted]

(Ex. 46) (emphasis added). While the European tribunal has not yet rendered a decision, and thus judicial estoppel is not yet technically applicable, IBM's position before the European Commission Competition Directorate-General is compelling extrinsic evidence supporting PSI's proffered construction. PSI therefore respectfully submits that its proposed construction of "firmware" should be adopted.

2. "firmware image" (claims 1, 9, 17)

IBM's Proposed Construction	PSI's Proposed Construction
Software copied from non-volatile memory and used to boot a partition.	An instance of a particular firmware.

To narrow this dispute, it is useful to start by noting that the parties do not appear to disagree about what an *image* is. An image is simply an "instance" or "copy" of the original. For example, the *Microsoft Computer Dictionary* defines an "image" in this context as "[a] duplicate, copy, or representation of all or part of a hard or floppy disk, a section of memory or

hard drive, a file, program, or data" (Ex. 30). Similarly, the *McGraw-Hill Dictionary of Scientific and Technical Terms* defines an "image" in this context as "[a] copy of the information contained in one medium recorded on a different data medium" (Ex. 43).

Thus, the only dispute between the parties with respect to "firmware image" is what "firmware" means. As noted above, IBM does not propose a construction for "firmware." Nevertheless, through its construction for "firmware image," IBM asks the Court to construe "firmware" as ***any*** "non-volatile memory" and a "firmware image" as a copy of any such firmware.

IBM's construction should be rejected because it is contrary to both the intrinsic and extrinsic evidence. First, IBM's construction is improper because it includes software stored in ***any*** non-volatile medium (*i.e.*, any storage device whose contents are not lost when power is cut off). Thus, under IBM's construction, a "firmware image" might be software copied from a floppy disk, a CD-ROM, a hard drive, or even an ipod. But, as noted in the previous section, the '002 specification expressly defines "firmware" as being software stored in non-volatile memory ***chips***, not software stored in ***any*** non-volatile memory. '002 patent at 6:22-24. Because the '002 patentee acted as lexicographer with respect to "firmware," that definition (which is PSI's construction) should be adopted.

Second, IBM's construction would destroy the very concept of firmware. The whole point of *firm*ware is to protect information by storing it in a location that is generally *inaccessible* to the user and the operating system, and therefore protected from erasure. By expanding the concept of firmware to include software stored on any non-volatile media, ***including*** non-volatile media (like a floppy disk) ***that can be overwritten by any program or application***, IBM's proposal runs contrary to the purpose of firmware.

Third, IBM itself uses the term "firmware image" to refer to information that is copied from non-volatile memory ***chips***, and not to refer generally to any "software copied from non-volatile memory." For example, in the System p5 590 and 595 Technical Overview and Introduction, IBM states that "[d]uring boot time the system service processor dynamically allocates the firmware image from flash memory into main system memory." (Ex. 44 at 24). "Flash memory," according to IBM, is "[a] ***computer chip*** with a read-only memory that retains its data when the power is turned off and that can be electronically erased and reprogrammed without being removed from the circuit board.." (Ex. 49).

Similarly, another IBM manual entitled "L10 Implementation" instructs users how to "update the firmware image" by performing a "ROM update." (Ex. 45 at 22-26). ROM, or "read-only memory," is a type of silicon-based memory ***chip*** that holds its content without electrical power. Indeed, it is defined in the *Microsoft Computer Dictionary* as a "semiconductor circuit into which code or data is permanently installed by the manufacturing process" (Ex. 30).

Finally, IBM argues that the "key feature" of a "firmware image" is that "it is used to boot the partition." This argument is a distraction. The '002 patent obviously uses firmware images to boot the partition. But noting how a firmware image is ***used*** in one context does not tell you what a firmware image actually ***is***. For example, the '261 patent uses a "processor" to perform a specific type of emulation, while the '709 patent uses a "processor" to round floating point numbers—and yet the parties agree that "processor" has the same definition unrelated to these tasks in both patents (as well as in the '495, '520, '789 and '678 patents, all of which use "processors" for their own distinct tasks, too). It is beyond dispute that firmware and firmware images can be used for tasks other than booting a partition,[40] and there is no ***reason*** to define "firmware image" by this specific use—particularly where, as here, the claims themselves

[40] For example, the firmware image referred to in the "L10 Implementation" document cited above is used in the operation of a Storage Area Network switch. *See* www.redbooks.ibm.com/redpapers/pdfs/redp4178.pdf.

expressly state how the firmware image is to be used. IBM's "key feature" argument is simply an attempt to run from the fact that the '002 specification expressly defines "firmware," and that this intrinsic definition is contrary to the construction IBM wants.

In light of the intrinsic evidence defining "firmware," and the lack of any reason to contravene that intrinsic evidence, PSI's construction should be adopted.

3. "storing a plurality of different firmware images in said computer system/a plurality of different firmware images being stored in said computer system" (claims 1, 9, and 17)

IBM's Proposed Construction	PSI's Proposed Construction
Storing more than one distinct firmware image in the computer system.	Storing at least two firmware images that are not the same as each other.

IBM has agreed to accept PSI's construction. *See* IBM Br. at 81.

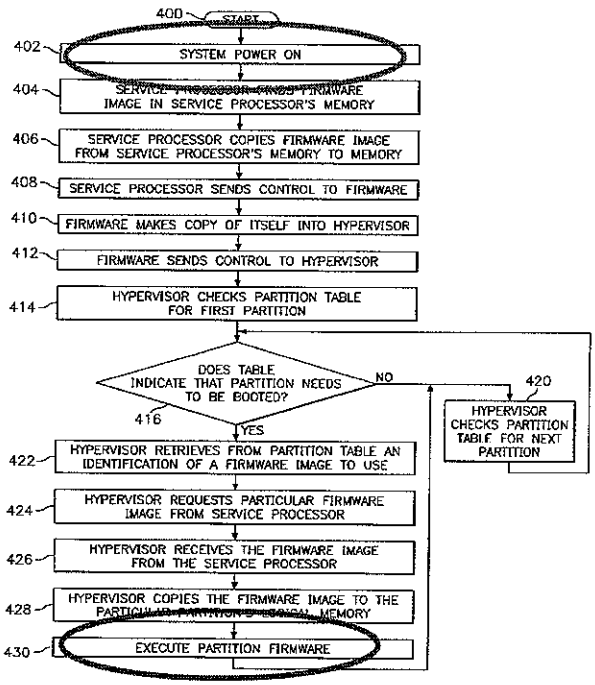
4. "capable of being executed during a power-on process to boot said computer system" (claims 1, 9, 17)

IBM's Proposed Construction	PSI's Proposed Construction
Capable of being executed during the operations performed by a computer system from the time it is turned on until it is ready to run applications to boot said computer system.	Able to be executed during the operations performed by a computer system from the time it is turned on until it begins executing the partition firmware.

The only dispute between the parties with respect to this claim limitation is *when* the "power on process" *ends*. IBM contends that the "power on process" lasts until the computer system is "ready to run applications to boot [the] computer system," while PSI believes that the power-on process ends when the partition begins executing the partition firmware.

Although this seems like a highly technical dispute, the specification explicitly ***shows*** what it means by a "power on process to boot the computer system" in Figure 4. Specifically, it is clear that Figure 4 illustrates "booting" the system as called out in the claims. As the specification puts it, Figure 4 "depicts ***booting*** one partition using one of a plurality of different firmware images maintained by a logically partitioned computer system." '002 patent at 2:38-42.

Nor is there any doubt that Figure 4 depicts the "power-on process," since one of its steps is "SYSTEM POWER ON":



Because Figure 4 depicts the "power on" process of "booting" the computer system, the last step in that Figure is, by definition, the last step in the process. Yet, Figure 4 ends not with the system being "ready to run applications," as IBM would have it, but instead with the step labeled 430, and circled in red above. That step is "EXECUTE PARTITION FIRMWARE." Thus, the power-on process as disclosed in the patent specification ends with the execution of the partition firmware, exactly as PSI contends.

That the "power-on process" ends with the execution of the partition firmware rather than when the computer "is ready to run applications," as IBM erroneously contends, is also mandated by the language of the claims. The claims require "rebooting said one of said plurality of partitions utilizing said one of said plurality of firmware images ***prior to booting an operating system*** in said one of said plurality of partitions." '002 patent at 8:41-45 (claim 1). It is well known in the art that, (with only a few exceptions not applicable to the mainframes at issue in the

patent) computer systems run applications *after* the operating system is up and running. *See* Patt Decl. ¶ 40-41. This is because, according to the *Microsoft Computer Dictionary*, "the operating system is the foundation software on which applications depend" (Ex. 30 at 378, definition of "operating system"). Because, as a practical matter, normal applications cannot run before an operating system has started, a person of ordinary skill in the art would interpret the claim language "prior to booting an operating system" to mean that the power-on process ends before applications can be run. *See* Patt Decl. ¶ 41 It is thus a logical contradiction to say, as IBM does, that the "power-on process" lasts until the computer "is ready to run applications." The claims, instead, require the process to end "prior to booting an operating system," which must (in turn) be prior to the point at which "applications" can be run. To put it another way, the "power on process to boot the computer system" must (according to the claims) be performed "prior to booting an operating system," which must in turn be performed before the computer is "ready to run applications." Thus, it is wrong to say, as IBM does, that the power on process lasts until the computer is ready to run applications.

IBM's construction is irreconcilable with the claim language and the specification. PSI's construction, which is taken directly from the intrinsic evidence, should be adopted.

5. "instruction means for storing a plurality of different firmware images in said computer system" (claim 9)

IBM's Proposed Construction	PSI's Proposed Construction
Function: Instructing the computer to store a plurality of different firmware images in the computer system. Structure: A ***computer program product*** stored in a computer recordable-type media (***see, e.g.***, 8:4-20) that instructs a computer to store a plurality of different firmware images in the computer system (***see, e.g.***, Fig. 3, 6:13-16), and equivalents thereof.	Function: Storing a plurality of different firmware images in said computer system. Structure: No corresponding structure.

IBM's proposal is facially inadequate. It attempts to claim ***all*** programming that causes a general purpose computer to perform the function recited in the claim. But, under *Aristocrat*, a means-plus-function limitation cannot be construed in this manner. *Aristocrat*, 521 F.3d at 1331 ("Because general purpose computers can be programmed to perform very different tasks in very different ways, simply disclosing a computer as the structure designated to perform a particular function does not limit the scope of the claim...as required by section 112 paragraph 6").

Furthermore, as with the means-plus-function limitations found in the '812 patent, the passage IBM identifies as a purported ***example*** of corresponding structure is nothing more than boilerplate. That passage, in its entirety, says only that:

> It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media, such as a floppy disk, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs, and transmission-type media, such as digital and analog communications links, wired or wireless communications links using transmission forms, such as, for example, radio frequency and light wave transmissions. The computer readable media may take the form of coded formats that are decoded for actual use in a particular data processing system.

'002 patent at 8:4-20. This passage says nothing at all about the particular algorithm needed to accomplish the claimed function, and consists of no more than a generic description that applies equally to any computer software. *Aristocrat* squarely establishes that this is inadequate. *Aristocrat*, 521 F.3d at 1328.

Indeed, ***IBM has used this <u>same</u> generic language in <u>over 500</u> other patents.***[41] This boilerplate language clearly cannot be construed as providing adequate corresponding structure for the specific function claimed here.

---

[41] *See* http://www.google.com/patents?lr=&q=%22It+is+important+to+note+that+while+the+present+invention+has+been+described+in+the+context+of+a+fully+functioning+data+processing+system%22 (listing patents).

The only other "example" of corresponding structure IBM cites, "Fig. 3, 6:13-16," is also inadequate. Figure 3 is nothing more than a generic diagram of the parts of a partitioned mainframe, as the specification itself tell us: "Fig. 3 is a block diagram of an exemplary logically partitioned platform in which the present invention may be implemented." *Id.* at 2:35-37. This block diagram fails to disclose any specific structure or algorithm for actually carrying out the "storing a plurality of firmware images" function and is therefore no more capable of constituting corresponding structure that the boilerplate language cited above.

The language in column 6 is also inadequate, since it does nothing more than ***describe*** the claimed function: "The partition tables and firmware images described herein, as well as other information, are stored within service processor memory 291." '002 patent at 6:13-16. Language that simply ***describes*** the function cannot constitute corresponding structure. *Aristocrat*, 521 F.3d at 1334 (holding that "language [that] simply describes the function to be performed" is insufficient to constitute corresponding structure under § 112 ¶ 6).

IBM fails to identify any portions of the specification that constitute adequate corresponding structure, and it relies principally on a passage that is so generic that it has been used verbatim as boilerplate in over 500 other IBM patents. PSI's construction should be adopted and the claim invalidated for failure to comply with section 112.

6. "instruction means for rebooting one of said plurality of partitions utilizing one of said plurality of firmware images without rebooting other ones of said plurality of partitions" (claim 9)

IBM's Proposed Construction	PSI's Proposed Construction
Function: Instructing a computer to reboot one of the plurality of partitions utilizing one of the plurality of firmware images without rebooting other partitions. Structure: A ***computer program product*** stored in a computer recordable-type media (***see, e.g.***, 8:4-20) that instructs a	Function: Rebooting one of said plurality of partitions utilizing one of said plurality of firmware images without rebooting other ones of said plurality of partitions. Structure: No corresponding structure.

IBM's Proposed Construction	PSI's Proposed Construction
computer to reboot one of the plurality of partitions utilizing one of the plurality of firmware images without rebooting other partitions (***see, e.g.***, Fig. 3, 6:58-67; **or** blocks 422-430 of Fig. 4, 7:30-45), and equivalents thereof.	

Here, again, IBM seeks to claim as corresponding structure ***any*** "computer program" that performs the function recited in the claims, and it cites the same general boilerplate language (at 8:4-20) that appears in hundreds of other IBM patents. The arguments above apply with equal force here.

The description of the *function* performed by this generic "computer program" is also insufficient to constitute an algorithm. IBM cites two passages which it says describe the generic software's performance of the function of "instruct[ing] a computer to reboot one of the plurality of partitions utilizing one of the plurality of firmware images without rebooting other partitions." Each, however, fails to meet the standard set forth in *Aristocrat,* as set forth below.

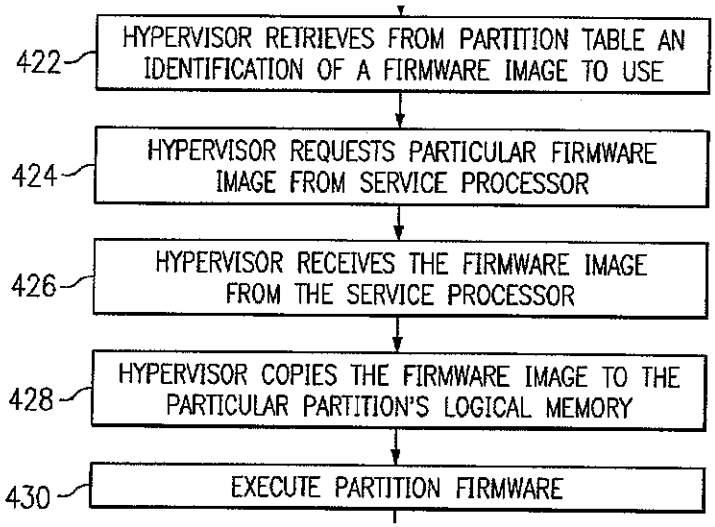
First, IBM cites Figure 3, which depicts the general layout of the mainframe computer. But as noted above, a general purpose computer *cannot* be a corresponding structure under *Aristocrat*. 512 F.3d at 1334 ("In cases involving a computer-implemented invention in which the inventor has invoked means-plus-function claiming, this court has consistently required that the structure disclosed in the specification be more than simply a general purpose computer or microprocessor.").

Next, IBM cites "6:58-67," which states the following:

> As described in more detail below, hypervisor 210 routinely checks each partition table to determine a current state of the indicator stored in each table. When hypervisor 210 finds an indicator that indicates a partition needs to be rebooted, hypervisor 210 copies the firmware image identified within that partition table to the logical memory of the partition associated with the partition table. That firmware image is then executed within the partition causing only that partition to be rebooted. Other partitions are unaffected by this process.

This passage does not provide sufficient detail to constitute a structure. It does not describe the location of the "partition table," or say ***how*** it is to be "check[ed]." It does not say how often checking must be in order to be done "routinely," or in what manner the "indicator that indicates a partition needs to be rebooted" is stored. It does not say how to locate the "firmware image identified within that partition table," or where such a firmware image might have come from. Nor does it say ***how*** the computer is to "execute[]" the firmware image within the partition. This passage is nothing more than a ***description*** of the function; it does not constitute corresponding structure because it does not identify the ***algorithm*** used to ***actually achieve*** the general phases of the rebooting process the process describes. *See Aristocrat*, 521 F.3d at 1334 (holding that "language [that] simply describes the function to be performed" is insufficient to constitute corresponding structure under § 112 ¶ 6). IBM's contention that this passage is corresponding structure is like saying that the sentence "build a rocket, load the astronauts and send them to Mars" is an *algorithm* for conducting a NASA expedition. That sentence recites a series of steps, but it cannot qualify as corresponding structure because it does not "particularly point out and distinctly claim the subject matter which the applicant regards as his invention." 35 U.S.C. § 112 ¶ 2.

Finally, IBM cites "blocks 422-430 of Fig. 4, 7:30-45," which are the following:



> Referring again to block 416, if a determination is made that the partition associated with the partition table does need to be rebooted, the process passes to block 422 which illustrates hypervisor 210 retrieving from the partition table an identification of one of multiple, different firmware images to use. Next, block 424 depicts hypervisor 210 requesting the firmware image identified by the identification retrieved from the partition table.

This figure and passage describe exactly the same set of general phases of the rebooting process that are described in the first passage IBM cited, discussed above. As above, the location and structure of the referenced "hypervisor," "partition table," and "identification" are not specified. Similarly, the steps needed to "retriev[e]" that "identification" from the "partition table" and then "execute" the partition firmware are omitted. Far from a specific algorithm, as *Aristocrat* requires, the specification gives only the general outlines of the phases in the rebooting process.

For these reasons, IBM's purported "examples" of corresponding structure are inadequate.

VIII. THE '851 I/O PATENT

As noted above, mainframes are frequently divided into several "partitions," each of which acts as if it were an independent computer. Sometimes it is desirable to share input/output ("I/O") devices, such as disk drives, among partitions. To ease integration with existing programs, IBM devised a system by which several partitions can share a single disk drive

without each partition having to know that the disk drive is being shared. As far as each partition is concerned, the shared hard drive is not shared at all.

The system described in the '851 patent accomplishes this by presenting a separate "image" of the shared resource to each partition. Each partition has its own "image" of each shared resource, and each "image" is represented by a hardware or microprogramming construct called an "input/output control block." *See* '851 patent at 7:43-60.

The parties dispute the construction of the term "input/output control blocks." IBM contends that the term refers to any data structure which contains information about any input/output resource. PSI's construction, on the other hand, limits the meaning of that term to that which the inventors of the '851 identified as their "invention" in the specification.

1. "input/output control blocks" (claims 9, 21, 22)

IBM's Proposed Construction	PSI's Proposed Construction
Data structures containing information about I/O resources.	A hardware or microprogramming construct which specifies a shared resource to an OS, and may be said to represent an image of the resource to each sharing OS.

PSI's construction is the combination of two statements in the specification about "input/output control blocks." First, the specification states that an I/O control block is a "hardware or microprogramming construct":

> Each image of each channel, subchannel, or logical control unit is represented in the I/O subsystem by use of a ***hardware or micro-programming constructs***, herein called "channel control blocks" (CHCBs), "sharable subchannel control blocks" (SSCBs), and "logical control unit control blocks" (LCUCBs), respectively. The CHCBs, SSCBs, and LCUCBs are all located in the I/O subsystem storage of the CEC.

Id. at 7:61-68 (emphasis added). CHCBs, SSCB, and LCUCBs are all examples of I/O control blocks. *See, e.g.*, 34:58-60 (referring to "a sharing set ***of I/O control blocks (CHCBs)*** for each of a plurality of physical I/O channels").

Second, the specification states that an input/output control block, as PSI contends, "specifies a shared resource to an OS, and may be said to represent an image of the resource to each sharing OS":

> ***The invention*** provides a novel method of sharing I/O channels, control units, and devices by a number of different OSs by physically providing multiple control blocks for the respective use by the OSs, each of which ***specifies a shared resource to an OS, and may be said to represent an image of the resource to each sharing OS***.

Id. at 7:43-49 (emphasis added). IBM should be held to these descriptions of "[t]he invention," since "the scope and outer boundary of claims is set by the patentee's description of his invention." *On Demand Mach. Corp. v. Ingram Indus., Inc.*, 442 F.3d 1331, 1338 (Fed. Cir. 2006).

The Federal Circuit's decision in *Honeywell Int'l, Inc. v. ITT Industries, Inc.*, 452 F.3d 1312 (Fed. Cir. 2006), is particularly instructive here. In *Honeywell* the district court construed a "fuel injection system component" to mean "a fuel filter," notwithstanding that given its ordinary meaning the term "refers to any constituent part of the fuel injection system of a motor vehicle including, for example, fuel filters, fuel lines, and connectors." *Honeywell*, 452 F.3d at 1315-16. Indeed, the applicants had made statements in the prosecution history suggesting that the term "fuel injection system component" was broader than only fuel filters. *Id.* at 1316. However, in the written description, "the 'invention' was identified to be only a fuel filter." *Id.* Thus, "[g]iven the written description, the [district] court concluded that the patentee characterized a fuel filter as the only embodiment of his invention, not merely a preferred version of all possible embodiments." *Id.* (internal quotation omitted). The Federal Circuit affirmed, analyzing the issue in terms that apply precisely to this case:

> Here, the written description uses language that leads us to the conclusion that a fuel filter is the only 'fuel injection system component' that the claims cover, and that a fuel filter was not merely discussed as a preferred embodiment. On at least four occasions, the written description refers to the fuel filter as 'this invention' or

> 'the present invention'....***The public is entitled to take the patentee at his word and the word was that the invention is a fuel filter.***

Id. at 1318. Here, just as in *Honeywell*, the public (and PSI) is entitled to take the '851 inventors at their word, and their word was that the invention provides a "hardware or microprogramming construct which specifies a shared resource to an OS, and may be said to represent an image of the resource to each sharing OS." 7:43-49,61-68.

In other cases, the Federal Circuit has similarly construed claims to be no broader than the "invention" as described in the specification. For example, in *SciMed Life Systems, Inc. v. Advanced Cardiovascular Systems, Inc.*, 242 F.3d 1337, 1338 (Fed. Cir. 2001), the claims recited a catheter. Two types of catheters were known in the art: dual and coaxial. *SciMed*, 242 F.3d at 1339. The Federal Circuit concluded that the term catheter should be construed as limited to a coaxial catheter because "the characterization of the coaxial configuration as part of the 'present invention' is strong evidence that the claims should not be read to encompass the opposite structure." *Id.* at 1343. *See also Watts v. XL Sys., Inc.*, 232 F.3d 877, 883 (Fed. Cir. 2000) (noting that the specification states that "the present invention" used the feature at issue); *Alloc, Inc. v. ITC*, 342 F.3d 1361, 1368-69 (Fed. Cir. 2003) (relying on the patent specification's description of "the invention").

IBM argues that these passages from the specification cannot be used to define the term "input/output control blocks" because they describe only a preferred embodiment of the claimed invention. But this is facially incorrect. The passages from which PSI takes its proffered construction appear in the section of the specification entitled "SUMMARY OF THE INVENTION." '851 patent at 6:26. *Cf. Microsoft Corp. v. Multi-Tech Sys., Inc.*, 357 F.3d 1340, 1348 (Fed. Cir. 2004) (construing claims to be consistent with the specification's "Summary of the Invention"). Later in the patent, particular embodiments of the invention are described in the section entitled "DESCRIPTION OF THE DETAILED EMBODIMENTS." '851 patent at

11:38-39. Indeed, the word "embodiment" first appears in the patent at column 8, line 7—***after*** the passages from which PSI takes its construction.[42]

Finally, the Court should note that IBM's construction encompasses not only the "input/output control blocks" described in the specification but also ***any*** other "data structure" which contains ***any*** sort of "information about I/O resources." This includes, for example, a PDF of a user's manual for a printer that one might store on a hard drive, since the PDF is a "data structure" and the printer user manual is "information about I/O resources." IBM's absurdly general construction also includes the entire contents of main memory, since the operating system must keep track of information about I/O resources at all times, and the data in main memory (like all data in a computer) fits within IBM's unbounded "data structure" construction. In short, it is hard to imagine anything that does not fall within the unbounded scope of IBM's construction.

The '851 claims and specification that were actually approved by the Patent Office do not reflect IBM's overly broad construction. Instead, the specification expressly describes the patent's "invention." PSI's construction is taken directly from that description and should be adopted.

IX. CONCLUSION

IBM's brief repeatedly acknowledges that it is "relying predominantly on extrinsic evidence" and repeatedly fails to cite even a single instance of intrinsic support for its proposed constructions. IBM Br. at 23 ("register"); *see also id.* at 17-19 ("instruction"), 54-55 ("machine instruction"); 62 ("internal dataflow"). Worse, the only extrinsic examples IBM could find to support its constructions for key terms like "processor" and "register" date to ***1964*** and ***1971***, more than 30 years prior to the time of the inventions. Smotherman ¶¶ 5-8, 15. IBM's

[42] *See* '851 patent at 8:5-9 ("The hypervisor may also be assigned a control block in a sharing set. In the preferred embodiment, IID=0 is assigned to the hypervisor, and the non-zero IIDs are assigned to OSs.")

constructions are so amorphous that they cannot be taken seriously—they turn a keyboard and a monitor into a "processor," allow a "floating point unit" to encompass a printer, and make a "converter" include things like Blackberries, cell phones and Russian-to-English translation software. The Court should reject IBM's borderline frivolous constructions and, instead, should adopt PSI's proposals, which rely principally on the intrinsic evidence.

Dated: June 5, 2008

SUSMAN GODFREY LLP

By: /s/ Tibor L. Nagy Jr.

Stephen D. Susman
Stephen Morrissey
Jacob Buchdahl
Tibor L. Nagy Jr.
Ryan Kirkpatrick
654 Madison Ave., 5th Floor
New York, NY 20065
(212) 336-8330

KEKER & VAN NEST LLP

Clement Roberts
Joseph Gratz
710 Sansome Street
San Francisco, CA 94111
(415) 391-5400